UNIVERSIDAD NACIONAL DE PIURA

FACULTAD DE INGENIERÍA INDUSTRIAL

ESPECIALIDAD DE INGENIERÍA INFORMÁTICA

"IMPLEMENTACIÓN DE MEJORAS EN EL FRAMEWORK DE DESARROLLO DE N-CAPAS ORIENTADO AL DOMINIO BASADAS EN TECNOLOGÍAS DSL PARA LA REDUCCIÓN DE LOS TIEMPOS DE DESARROLLO DE SOFTWARE"

PRESENTADO POR:

BC. NELSSON JOSÉ AGUILAR SALVADOR

ASESORADO POR:

ING. ARTURO SANDOVAL RIVERA

TESIS PARA OPTAR EL TÍTULO DE:

INGENIERO INFORMÁTICO

PIURA, PERU

JUNIO 2015

UNIVERSIDAD NACIONAL DE PIURA

FACULTAD DE INGENIERÍA INDUSTRIAL

ESPECIALIDAD DE INGENIERÍA INFORMÁTICA

"IMPLEMENTACIÓN DE MEJORAS EN EL FRAMEWORK DE DESARROLLO DE N-CAPAS ORIENTADO AL DOMINIO BASADAS EN TECNOLOGÍAS DSL PARA LA REDUCCIÓN DE LOS TIEMPOS DE DESARROLLO DE SOFTWARE"

TESIS PARA OPTAR EL TÍTULO DE:

INGENIERO INFORMÁTICO

TESISTA:

Bc. NELSSON JOSÉ AGUILAR SALVADOR

ASESOR:

Ing. ARTURO SANDOVAL RIVERA

PIURA, PERU

JUNIO 2015

UNIVERSIDAD NACIONAL DE PIURA
FACULTAD DE INGENIERÍA INDUSTRIAL
DECANATO

ACTA DE SUSTENTACIÓN DE TESIS

Los Miembros del Jurado Calificador Ad-Hoc de la Tesis denominada: **«IMPLEMENTACIÓN DE MEJORAS EN EL FRAMEWORK DE DESARROLLO DE N-CAPAS ORIENTADO AL DOMINIO BASADAS EN TECNOLOGÍAS DSL PARA REDUCCIÓN DE LOS TIEMPOS DE DESARROLLO DE SOFTWARE »,** presentada por el señor **NELSSON JOSÉ AGUILAR SALVADOR,** Bachiller de la Escuela Profesional en Ingeniería Informática; asesorada por **el Ing. Arturo Sandoval Rivera;** reunidos para la sustentación de ésta y luego de escuchar su exposición y las respuestas a las preguntas formuladas, la declaran:

Con el Calificativo: Aprobado

Excelente

En consecuencia el sustentante se encuentra apto para recibir el título profesional de **INGENIERO INFORMÁTICO**, conforme a Ley.

Piura, 20 de junio del 2015

Ing. PEDRO ANTONIO CRIOLLO GONZALES, MSc.
PRESIDENTE - JURADO CALIFICADOR

Ing. CARMEN ZULEMA QUITO RODRÍGUEZ, MSc.
VOCAL - JURADO CALIFICADOR

Ing. JORGE LUIS SANDOVAL RIVERA, MSc.
SECRETARIO - JURADO CALIFICADOR

Agradecimientos

A mi madre y a mi familia por la confianza depositada en mí.

A mis compañeros de la universidad de los cuales aprendí muchas de las cosas que hoy aplico en mi vida profesional.

A la Universidad Nacional de Piura por haberme inculcado la pasión por mi carrera.

RESUMEN

La presente investigación desarrolla la implementación de mejoras para el proceso de desarrollo de sistemas basados en tecnología Microsoft y que utilizan la arquitectura de N-Capas orientado al dominio propuesto en la guía del mismo nombre publicada por la corporación, estas mejoras se basan en la tecnologías Domain Specific Language y plantillas de proyecto Visual Studio Templates que se implementan como extensibilidad bajo el entorno de desarrollo Visual Studio 2012.

La finalidad de la investigación es desarrollar una herramienta que permita automatizar la generación de código para la capa de datos, capa de acceso a datos y la capa de dominio mediante el modelado de las entidades de dominio de la arquitectura de la Guía de N-Capas con el propósito de reducir los tiempos de desarrollo de un sistema que tenga una cantidad considerable de lógica de negocio, tanto en la fase de definición como en la fase de construcción.

La importancia de la investigación radica en la utilización de las herramientas brindadas por Microsoft a aquellas empresas que tienen una membresía Partner para poder tener un software de calidad en un menor tiempo.

El trabajo de investigación se desarrolló bajo la metodología Scrum para lo cual se utilizó el portal Microsoft Visual Studio Online y para el desarrollo de la herramienta se ha utilizado el IDE Visual Studio, utilizando una solución de extensibilidad que incluye un proyecto de lenguaje especifico de dominio DSL y plantillas de proyecto Visual Studio Templates, y como gestor de base de datos SQL Server 2012.

Palabras Clave

Framework: Marco de trabajo para el desarrollo de aplicaciones basado en un conjunto concreto de patrones de reutilización.

Arquitectura de N-Capas: Arquitectura propuesta por Microsoft Corporation y publicada en la guía oficial denominada "Guía de Arquitectura de N-Capas orientada al Dominio con .NET 4.0"

Domain Specific Language (DSL): Metalenguaje usado para modelar un dominio especifico de negocio y que se implementa bajo proyectos de extensibilidad de Visual Studio.

Scrum: Metodología ágil utilizada para el desarrollo de productos de software basada en los fundamentos de los principios ágiles.

ABSTRACT

This research develops the implementation of improvements to the process of development of systems based on Microsoft technology and using the N-tier domain oriented architecture proposed in the guide of the same name published by the corporation, these improvements are based on the technologies Domain Specific Language and Visual Studio project templates that are implemented as extensibility under the development environment Visual Studio 2012.

The purpose of the research is to develop a tool to automate code generation for the data layer, data access layer and domain layer by modeling domain entities architecture N-tier Guide in order to reduce development time of a system that has a substantial amount of business logic, both in the definition phase and the construction phase.

The importance of research lies in the use of the tools provided by Microsoft to companies that have a Partner membership to have a quality software in less time.

The research was developed under the Scrum methodology for which the Microsoft Visual Studio Online website was used and for the development of the tool was used Visual Studio IDE, using a solution of extensibility that includes a proposed domain specific language DSL and Visual Studio project templates, and as a database manager SQL Server 2012.

Keywords

Framework: Framework for the development of applications based on a specific set of patterns reuse.

N-Tier Architecture: Architecture proposed by Microsoft Corporation and published in the official guide called "Guía de Arquitectura de N-Capas orientada al Dominio con .NET 4.0"

Domain Specific Language (DSL): Metalanguage used to model a specific business domain and that is implemented under Visual Studio extensibility projects.

Scrum: Agile metodology used for the application development based in the fundaments of agile principies.

INDICE

Índice de Tablas

Índice de Figuras

INTRODUCCIÓN

La presente investigación consiste en la implementación de mejoras sobre el framework de desarrollo de aplicaciones empresariales que está basado en la arquitectura de N-Capas orientada al Dominio propuesta por Microsoft y plasmada en una guía publicada y dirigida a arquitectos y desarrolladores de software. Dichas mejoras se basan en la aplicación de la tecnología de metamodelado de objetos DSL (Domain Specific Language), con el fin de automatizar la generación de código mediante plantillas que utilizan un modelo de diseño gráfico de las entidades del dominio de negocio y generan el código necesario de acceso a datos para el Back-End de la aplicación de manera que se permita la reducción de tiempos de programación y desarrollo de sistemas que manejan una complejidad considerable en sus procesos de negocio.

La necesidad de desarrollar software de calidad basado en plataformas de Microsoft ha llevado a establecer normas y patrones de diseño de desarrollo de software reutilizables durante el desarrollo de software, dichos patrones y aspectos de la arquitectura se encuentran plasmados en una guía del arquitecto y desarrollador publicada por Microsoft Ibérica denominada *Guía Arquitectura N-Capas Orientada al Dominio*, dicha guía servirá como base de este proyecto de investigación para definir la arquitectura de la aplicación sobre la cual se implementará la tecnología de metamodelado propuesta.

Si bien la guía es un manual completo sobre toda la arquitectura y especifica patrones de diseño que van desde la capa de datos hasta la capa de presentación, esta investigación se centra en la parte de la arquitectura que define el núcleo de negocio de aplicación, el acceso y el repositorio de datos, es decir que durante el desarrollo de la investigación se implementará el modelador de las capas de entidad del dominio, datos (base de datos) y capa de acceso a datos, para lo cual se definirá una plantilla de proyecto basada en el framework propuesto.

El uso de la tecnología DSL para la presente propuesta se debe a la versatilidad con la que se puede aplicar a las distintas capas de la arquitectura y que además se basa en la reutilización de componentes, esta tecnología tiene precedentes de éxito en el diseño de la capa de presentación, para el modelado de interfaces de usuario, capa de servicios distribuidos, para el modelado de Web Services, y acceso a datos para la especificación y/o consumo del modelo de datos. Lo que se busca con esta investigación es brindarle al arquitecto y al desarrollador la herramienta base para el desarrollo de sistemas con un alto grado de calidad en el código y la reducción considerable en los tiempos de construcción del software.

La investigación se dividió en ocho capítulos. El primer capítulo se centra en la definición detallada del problema, los objetivos y la hipótesis de la investigación, en el capítulo segundo se habla sobre la tecnología y antecedentes base: Fundamentos de arquitectura de aplicaciones, la reutilización software mediante marcos de desarrollo o frameworks, patrones de diseño y generadores de código, en el capítulo tercero se define el tipo y diseño de la investigación. En el cuarto capítulo se recopila la información correspondiente a la tecnología, el capítulo quinto se identifican las mejoras a aplicar y en el capítulo sexto se diseñan las mejoras a aplicar, estas mejoras se desarrollan finalmente en el capítulo siete. El octavo y último capítulo se centra en las pruebas y finalmente se brindan las conclusiones y recomendaciones de la investigación.

CAPÍTULO I: PROBLEMA DE LA INVESTIGACIÓN

1.1. Descripción del Problema de la Investigación

Uno de los factores predominantes sobre la calidad del software es el tiempo de entrega del producto, la entrega en la fase de final o fase de transición cuando el software se construye bajo la metodología tradicional, o los entregables incrementales del producto en el caso de las metodologías ágiles.

En la mayoría de los proyectos de software los tiempos estimados no son certeros en su totalidad, esto se debe muchas veces a que a pesar del empirismo y la experiencia en desarrollo de software los tiempos estimados suelen moldearse de acuerdo a las fechas de entrega establecidas por el cliente. Esto da paso a una estimación que se aleja de la estimación real de cuánto tiempo debería demorar el desarrollo de una funcionalidad en particular y por consiguiente cuanto debería demorar el desarrollo del proyecto en su totalidad. Otras veces también se debe a que durante el desarrollo la funcionalidad puede cambiar con relación al giro del negocio o las funcionalidades ya inexistentes pueden extenderse como nuevas funcionalidades.

Como se mencionó anteriormente, independientemente de la estimación evaluada para proyecto, existen hitos marcados por el cliente para lo cual, muchas veces se tiene que reestablecer el cronograma y los tiempos de desarrollo planteados. Básicamente estos hitos se basan en la documentación y desarrollo del aplicativo que se está construyendo. En cuanto a la entrega de documentación, es incremental y depende siempre del avance del desarrollo del producto, por lo que, por implicancia, el tiempo de entrega del proyecto depende del desarrollo del aplicativo.

Actualmente la mayoría de empresas desarrolladoras que son partners de Microsoft, es decir que tienen una suscripción como colaboradores, o empresas que desarrollan y/o utilizan sistemas basados en esta plataforma usan frameworks y patrones de diseño arquitectónicos basados en capas, esta forma de desarrollo tiene bastante acogida y no solo es utilizado bajo esta plataforma sino por aquellas basadas en software libre debido al bajo acoplamiento que proporcionan.

Para aquellas empresas que trabajan bajo este concepto, existen varios factores que influyen durante el tiempo de desarrollo, algunos de estos factores son mayormente humanos, es decir, que el tiempo de desarrollo depende siempre del constructor del software, para el caso del inicio del proyecto depende del arquitecto definir una arquitectura sobre la cual se basa el desarrollo y durante el desarrollo del software depende del programador cuán rápido se avance con las funcionalidades definidas durante el análisis.

Se han identificado entonces los siguientes factores influyentes sobre el tiempo de desarrollo en las siguientes fases del proyecto:

Al inicio del proyecto

- Una vez definido el diseño de la arquitectura o framework de la aplicación sobre el que se va a trabajar, este se implementa manualmente sobre el IDE Visual Studio, es

decir, el arquitecto construye la estructura del proyecto agregando manualmente los proyectos y elementos de código necesarios para el proyecto.

Durante el desarrollo del proyecto

El desarrollo de los objetos sobre el framework es manual del lado del desarrollador, las clases y objetos de las siguientes capas se desarrollan manualmente.

- **Objetos de la Capa de Entidad del Dominio**
  - Entidades del dominio de negocio
  - Colecciones o listas de las entidades del dominio de negocio.
- **Objetos de la Capa de Datos**
  - Tablas
  - Procedimiento almacenado de inserción
  - Procedimiento almacenado de actualización
  - Procedimiento almacenado de eliminación
  - Procedimiento almacenado de consulta por identificador
  - Procedimiento almacenado de consulta
- **Objetos de la Capa de Acceso a Datos**
  - Clase de acceso a datos por entidad de dominio

Estos factores suelen ser repetitivos para cada proyecto de software y muchas veces resultan siendo monótonos para el desarrollador, entonces al ser iterativos surge la necesidad de evitar el trabajo repetitivo y de realizar tareas que pueden ser automatizadas mediante la generación de la arquitectura del proyecto y de la generación de código. Para ello el presente proyecto presentará una alternativa de desarrollo basado en la guía propuesta por Microsoft para el desarrollo de aplicaciones empresariales a las cuales se agregará tecnología DSL (Lenguaje Especifico de Dominio) basada en el metamodelado visual del dominio de las entidades del negocio para generación de código. Al framework de desarrollo lo llamaremos framework N-Capas orientado al dominio, debido a la guía de desarrollo publicada por Microsoft con el mismo nombre, y se definirá la forma en la que la aplicación de esta tecnología influye en el tiempo de desarrollo del aplicativo a desarrollar.

1.2. Formulación del Problema de la Investigación

¿Se pueden reducir los tiempos de desarrollo de un aplicativo implementando mejoras basadas en DSL sobre el framework N-Capas orientado al dominio mejorando la productividad de los desarrolladores?

1.3. Justificación, Importancia y Beneficiarios de la Investigación

Durante el diseño y desarrollo del software se suelen identificar características que pueden resultar monótonas para el desarrollador como la creación manual de objetos de las diferentes capas de la aplicación como clases de lógica de negocio, entidades de negocio, acceso a datos, etc. Si bien se han desarrollado generadores de código y modeladores de diseño sobre las diferentes capas como Entity Framework para el acceso a datos, Web Service Software Factory para modelar web services y diseñadores de interfaces para clientes como el motor de vistas ASP, cada una de estas cumple una

función específica y están orientadas cada una a cubrir una necesidad específica de la arquitectura.

Si bien se pueden utilizar estas herramientas para diseñar la arquitectura de aplicación, estas no se ajustan a un modelo de arquitectura simple y a la vez flexible orientado a capas que utilizan las empresas de desarrollo que siguen los patrones de la guía propuesta por Microsoft. La solución a elaborar se enfoca en implementar en el framework de N-Capas orientado al dominio aplicando tecnología DSL, un modelador propio que cubra las características que se requieren en el desarrollo de aplicativos y que hoy son procesos manuales, con el fin de disminuir los tiempos de desarrollo de aplicación en cuanto a la capa entidades del dominio de negocio de la aplicación y su interacción con el repositorio de datos.

Al implementar esta solución se espera disminuir los tiempos de desarrollo de cada uno de los proyectos que usen tecnología Microsoft, beneficiando a:

- ✓ **Las empresas desarrolladoras de software** dado que al disminuir el tiempo de desarrollo se van a mejorar los tiempos de entrega a los clientes.

- ✓ **Los desarrolladores de software**, dado que las tareas repetitivas se automatizarán y mejorará la productividad de los mismos.

1.4. Objetivos de la Investigación

1.4.1. Objetivo General

- ✓ Implementar mejoras en el framework de N-Capas orientado al Dominio de la plataforma Microsoft que permita reducir los tiempos de desarrollo de software.

1.4.2. Objetivos Específicos

- ✓ Establecer la arquitectura sobre las mejoras en el framework de N-Capas orientado al Dominio de la plataforma Microsoft.
- ✓ Establecer el diseño de las mejoras en el framework de N-Capas orientado al Dominio de la plataforma Microsoft.
- ✓ Implementar mejoras en el framework de N-Capas orientado al Dominio de la plataforma Microsoft que permita aumentar la productividad de los desarrolladores.
- ✓ Implementar la generación de código para la capa de entidad en el framework de N-Capas orientado al Dominio de la plataforma Microsoft.
- ✓ Implementar la generación de código para la capa de datos en el framework de N-Capas orientado al Dominio de la plataforma Microsoft.
- ✓ Implementar la generación de código para la capa de acceso a datos en el framework de N-Capas orientado al Dominio de la plataforma Microsoft.
- ✓ Elaborar la documentación sobre el uso de las mejoras en el framework de N-Capas orientado al Dominio de la plataforma Microsoft.

1.5. Hipótesis de la Investigación

Hipótesis de la Investigación:

Hi: El uso de las mejoras en el framework N-Capas orientado al dominio permitirá reducir los tiempos de desarrollo del aplicativo.

Hipótesis Nula:

Ho: El uso de las mejoras en el framework N-Capas orientado al dominio no permitirá reducir los tiempos de desarrollo del aplicativo

Hipótesis Alternativas:

H1: El uso de las mejoras en el framework N-Capas orientado al dominio permitirá aumentar la productividad de los desarrolladores.

1.5.1. Identificación y Operacionalización de Variables

Identificación de Variables

Variable Independiente:

- ✓ Mejoras en el framework de N-Capas orientado al dominio.

Variable dependiente:

- ✓ Mejora en el tiempo de entrega de desarrollo de aplicativos, haciendo uso de las mejoras en el framework de N-Capas orientado al dominio.

Operacionalización de Variables

Variable	Definición Conceptual	Definición Operacional	Dimensiones	Indicadores
Mejoras en el framework de N-Capas orientado al dominio	Se implementará las mejoras basadas en DSL sobre las siguientes capas del Framework: - Estructura de la solución. - Componente de Entidad. - Componente de Base de Datos. - Componente de Capa de Acceso a Datos.	Con el uso en las mejoras se eliminará las tareas repetitivas de las siguiente manera: - Se creará la arquitectura de la solución mediante el uso de plantillas. - Se generará código para los componentes: Entidad, Base de Datos y Acceso a datos.	Funcionalidad del Framework mejorado.	✓ Cumplimiento de los requerimientos de mejoras. ✓ Integración de las mejoras con el framework de N-Capas.
Mejora en el tiempo de entrega de desarrollo de aplicativos, haciendo uso de las mejoras en el framework de N-Capas orientado al dominio.	Automatización de las tareas repetitivas.	Gestión de los tiempos de desarrollo haciendo uso del framework mejorado.	Gestión de carga de trabajo.	✓ Determinar la mejora en los tiempos de entrega. - Tiempos de entrega del componente con framework. - Tiempos de entrega del componente sin framework. ✓ Determinar la mejora en la productividad en los desarrolladores.

Tabla 1.5.1. Operacionalización de Variables
Fuente: Elaboración Propia

CAPÍTULO II: REVISIÓN DE LA LITERATURA O MARCO TEÓRICO

2.1. Antecedentes de la Investigación

En la Universidad de Utrecht, Sander Mak (2007) realizó un trabajo de investigación en el cual presenta un estudio objetivo sobre los lenguajes específicos del dominio desde el punto de vista del desarrollo de software basado en modelos (MDSD), haciendo hincapié en la mejora sobre los tiempos de respuesta de desarrollo de software que su implementación supone. Uno de los puntos de vista del trabajo de investigación es la modularización de lenguajes de alto nivel que permite tener un enfoque de interacción durante el desarrollo y evalúa cual sería la mejor forma de elegir un DSL para un proyecto determinado basándose en esta modularización. Este trabajo de investigación se centra en la evaluación de un lenguaje específico de dominio para la capa de acceso a datos y la capa de presentación de un framework denominado Erlang aplicado a tecnología de software libre, entre sus conclusiones sobre la evaluación Sander especifica el uso de este DSL para proyectos pequeños por su diseño muy estático.

Por otro lado Jeroen Dobbe (2006), de la Universidad Tecnológica Delft (Holanda) centra su trabajo de investigación en el desarrollo de lenguajes específicos de dominio a un nivel monolítico (no modularizado), es decir orientado específicamente a cubrir exclusivamente el modelado de alto nivel de software en la industria de los videojuegos. El trabajo de investigación de Dobbe centra su tema en la potencialidad que tiene un lenguaje DSL implementado en librerías denominadas Caníbal Engine para el facilitamiento y reducción de recursos en el desarrollo de software, concluyendo que sería de mucha utilidad el uso de este concepto para implementar videojuegos sin ir más allá de conocer el detalle técnico de los lenguajes y herramientas gráficas convencionales (lenguajes de propósito general y motores gráficos) si se usaran los DSL para la creación de software de entretenimiento.

Kristoffer Rosén (2013) en un trabajo más reciente en la Universidad de Lund (Suecia) plantea el uso de los lenguajes de alta abstracción para el desarrollo de software bajo plataformas móviles y smartphones, Rosén propone el uso de lenguajes DSL para la implementación de aplicaciones móviles en diferentes plataformas (IOS de Apple y Android), debido a la complejidad que puede suponer desarrollar una misma aplicación para estos dos sistemas operativos. El trabajo de investigación se basa en que se puede desarrollar un lenguaje de metamodelado para definir la estructura de la aplicación y generar código nativo para ambas plataformas. Haciendo uso del Framework Spoofax, Rosén demuestra que se puede definir reglas de gramática en un lenguaje específico de dominio con el fin de producir código nativo para Android y Iphone, concluyendo que los DSL facilitan el desarrollo de software multiplataforma.

2.2. Marco Conceptual

Fundamentos y Definición de la Arquitectura de Aplicaciones

En la etapa inicial del desarrollo de proyectos de software se requiere el análisis de requerimientos por parte de un arquitecto de software con el fin de definir la arquitectura que se va a utilizar para implementar la solución de negocios que cubrirá con las necesidades especificadas por los usuarios. En una definición de lo que es la arquitectura de aplicaciones la guía de la Microsoft denominada *"Guía de arquitectura de N-Capas orientada al Dominio"* a la cual de ahora en adelante llamaremos Guía de N-Capas o simplemente Guía Microsoft, sobre la cual se basará el presente proyecto de investigación, describe el diseño de la arquitectura de la siguiente manera:

El diseño de la arquitectura de un sistema es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes forman el sistema, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados como seguridad, disponibilidad, eficiencia o usabilidad. (De La Torre Llorente, 2010)

Según la Guía de N-Capas este paso es fundamental ya que esto definirá el éxito o fracaso del proyecto en sí mismo.

Durante el diseño de la arquitectura se tratan los temas que pueden tener un impacto importante en el éxito o fracaso de nuestro sistema. (De La Torre Llorente, 2010)

Así mismo define una serie de preguntas a responderse antes de tomar decisiones en cuanto a la definición de la arquitectura de aplicaciones.

- ¿En qué entorno va a ser desplegado nuestro sistema?
- ¿Cómo va a ser nuestro sistema puesto en producción?
- ¿Cómo van a utilizar los usuarios nuestro sistema?
- ¿Qué otros requisitos debe cumplir el sistema? (seguridad, rendimiento, concurrencia, configuración...)
- ¿Qué cambios en la arquitectura pueden impactar al sistema ahora o una vez desplegado?

Algunas de estas preguntas son la base de la elección de la solución a brindar en el presente proyecto de investigación tomando en cuenta los intereses de los distintos agentes que participan en el proceso de definición de la arquitectura del software. Como objetivo de este paso la Guía Microsoft define lo que debería especificar la arquitectura planteada para la solución de negocios brindada al cliente.

En síntesis, la arquitectura debería:

- Mostrar la estructura del sistema pero ocultar los detalles.
- Realizar todos los casos de uso.
- Satisfacer en la medida de lo posible los intereses de los agentes.
- Ocuparse de los requisitos funcionales y de calidad.
- Determinar el tipo de sistema a desarrollar.

- Determinar los estilos arquitecturales que se usarán.
- Tratar las principales cuestiones transversales.

Decisiones de Diseño Arquitectónico

De la misma forma Sommerville en su publicación de Ingeniería de Software define el diseño arquitectónico como resultado inicial del análisis de la estructuración de los sistemas de negocios en diversos subsistemas que los componen y la relación que estos mantienen entre sí como parte del todo.

Los grandes sistemas siempre se descomponen en subsistemas que proporcionan algún conjunto de servicios relacionados. El proceso de diseño inicial que identifica estos subsistemas y establece un marco para el control y comunicación de los subsistemas se llama diseño arquitectónico. El proceso de diseño arquitectónico está relacionado con el establecimiento de un marco estructural básico que identifica los principales componentes de un sistema y las comunicaciones entre estos componentes. (Sommerville, 2005)

En la misma publicación Sommerville señala las ventajas de diseñar y documentar la arquitectura del software.

1. Comunicación con los stakeholders. La arquitectura constituye una presentación de alto nivel del sistema que puede usarse como punto de discusión por varios stakeholders.
2. Análisis del sistema. Hacer explícita la arquitectura del sistema en una etapa temprana del desarrollo del sistema requiere realizar algún análisis. Las decisiones de diseño arquitectónico tienen un gran efecto sobre si el sistema puede cumplir los requerimientos críticos (tales como rendimiento, fiabilidad y mantenibilidad.
3. Reutilización a gran escala. Un modelo de arquitectura del sistema es una descripción compacta y manejable de cómo se organiza un sistema y cómo interoperan sus componentes. (Bass et al., 2003)

La definición y documentación de la arquitectura del aplicativo es entonces el punto de partida de la construcción del software y para las empresas desarrolladoras es a veces el punto de partida del desarrollo de todos los aplicativos a desarrollar debido a que este diseño se utilizará para los diversos proyectos de software a implementar a futuro.

La arquitectura del sistema es a menudo la misma para sistemas con requerimientos similares y, por lo tanto, pueden soportar reutilización del software a gran escala. Es posible desarrollar arquitecturas para líneas de productos en donde la misma arquitectura se usa en varios sistemas relacionados. (Sommerville, 2005)

Hofmeister y otros (Hofmeister et al., 2000) ponen de manifiesto cómo las etapas del diseño arquitectónico fuerzan a los diseñadores del software a considerar aspectos de diseño claves en etapas tempranas del proceso. Sugieren que la arquitectura del software puede servir como un plan de diseño que se usa para negociar los requerimientos del sistema y como una forma de estructurar las discusiones con los clientes, desarrolladores y gestores. También sugieren que es una herramienta esencial para la gestión de la complejidad. La arquitectura del software oculta detalles y permite a los diseñadores centrarse en las abstracciones clave del sistema. (Sommerville, 2005)

Las decisiones del arquitecto de software con respecto a la arquitectura de software definen el grado de abstracción que tiene el diseño, y es el grado de abstracción es el que define el nivel de reutilización del diseño arquitectural y de los componentes que la integran.

El proceso de diseño en la mayoría de las disciplinas de ingeniería se basa en la reutilización de sistemas o componentes existentes. Los ingenieros mecánicos o eléctricos no especifican normalmente un diseño en el que cada componente tenga que ser fabricado de una forma especial. Basan su diseño en componentes que han sido utilizados y probados en otros sistemas. Estos componentes no son solamente pequeños componentes, como tuercas y válvulas sino que incluyen subsistemas mayores, como motores, condensadores o turbinas. (Sommerville, 2005)

Reutilización del Software

Existen diversos niveles de reutilización de componentes de software, estos niveles son definidos como unidades de software por Sommerville son clasificados en su publicación de la siguiente manera.

1. **Reutilización de sistemas de aplicaciones.** La totalidad de un sistema de aplicaciones puede ser reutilizada incorporándolo sin ningún cambio en otros sistemas, configurando la aplicación para diferentes clientes o desarrollando familias de aplicaciones que tienen una arquitectura común pero que son adaptadas a clientes particulares.

2. **Reutilización de componentes.** La reutilización de componentes de una aplicación varía en tamaño desde subsistemas hasta objetos simples. Por ejemplo, un sistema de emparejamiento de patrones desarrollado como parte de un sistema de procesamiento de textos puede ser reutilizado en un sistema de gestión de base de datos.

3. **Reutilización de objetos y funciones.** Pueden reutilizarse componentes software que implementan una única función, como por ejemplo una función matemática o una clase de objetos. Esta forma de reutilización, basada en librerías estándar, ha sido habitual en los cuarenta últimos años. Están disponibles muchas librerías de funciones y clases para diferentes tipos de aplicaciones y plataformas de desarrollo. Éstas pueden utilizarse fácilmente enlazándolas con código de otras aplicaciones. En áreas como algoritmos matemáticos y gráficos, donde se necesita a un experto específico para desarrollar objetos y funciones, ésta es una aproximación particularmente efectiva.

Así mismo se definen las diversas aproximaciones sobre las cuales se aplica la reutilización de componentes de software.

Aproximaciones que soportan la reutilización del software.

Aproximación	Descripción
Patrones de diseño	Las abstracciones genéricas similares entre aplicaciones se representan como patrones de diseño que muestran los objetos abstractos y concretos y sus interacciones.
Desarrollo basado en componentes	Los sistemas se desarrollan integrando componentes (colecciones de objetos) que cumplen los estándares de modelado de componentes
Marcos de aplicaciones	Las colecciones de clases concretas y abstractas pueden adaptarse y extenderse para crear sistemas de aplicaciones.
Envoltura de sistemas heredados	Sistemas heredados (que pueden ser «envueltos» definiendo un conjunto de interfaces y proporcionando acceso a estos sistemas heredados a través de estas interfaces.
Sistemas orientados a servicios	Los sistemas se desarrollan enlazando servicios compartidos, que pueden ser proporcionados de forma externa.
Líneas de productos de aplicaciones	Un tipo de aplicación se generaliza alrededor de una arquitectura común para que pueda ser adaptada para diferentes clientes.
Integración COTS	Los sistemas se desarrollan integrando sistemas de aplicaciones existentes.
Aplicaciones verticales configurables	Un sistema genérico se diseña para que pueda configurarse para las necesidades de dientes de sistemas particulares.
Librerías de programas	Están disponibles para reutilización las librerías de funciones y de clases que implementan abstracciones comúnmente usadas.
Generadores de programas	Un sistema generador incluye conocimiento de un tipo de aplicación particular y puede generar sistemas o fragmentos de un sistema en ese dominio.
Desarrollo del software orientado a aspectos	Componentes compartidos entrelazados en una aplicación en diferentes lugares cuando se compila el programa.

Tabla 2.2.1. Aproximaciones que soportan la reutilización del software
Fuente: Ingeniería del Software (Sommerville, 2005)

Dada esta lista de técnicas para reutilización, la cuestión clave es ¿cuál es la técnica más adecuada a utilizar? Obviamente, esto depende de los requerimientos del sistema a desarrollar, la tecnología y activos reutilizables disponibles, y la experiencia del grupo de desarrollo. (Sommerville, 2005)

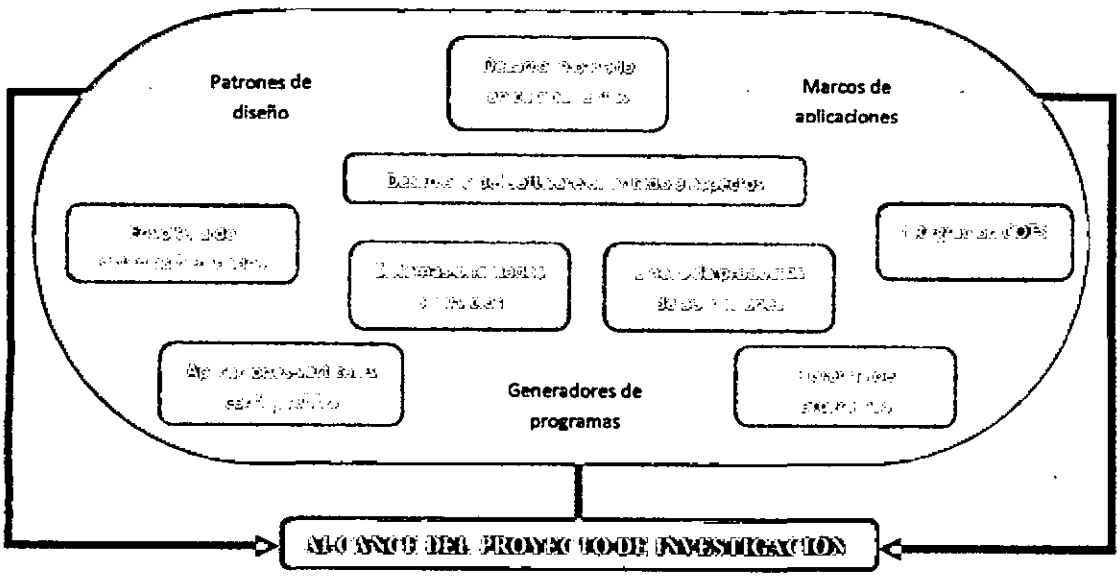


Figura 2.2.1. El campo de la reutilización
Fuente: Ingeniería del Software (Sommerville, 2005)

El presente proyecto de investigación se basa en aplicar estas aproximaciones de reutilización de software para definir una arquitectura que permita reducir los tiempos de desarrollo de aplicaciones, específicamente la solución planteada se centrará en las siguientes aproximaciones.

- Patrones de diseño
- Marcos de aplicaciones (frameworks)
- Generadores de programas

Patrones de Diseño

Cuando el diseñador intenta reutilizar componentes ejecutables, está limitado de forma inevitable por las decisiones de diseño detallado que han sido tomadas por los implementadores de esos componentes. Éstas varían desde algoritmos particulares que han sido utilizados para implementar los componentes hasta los objetos y tipos en las interfaces de los componentes. Cuando estas decisiones de diseño están en pugna con sus requerimientos particulares, reutilizar el componente es imposible o introduce ineficiencias en su sistema. Los patrones de diseño se derivaron de las ideas introducidas por Christopher Alexander (Alexander et al, 1977), quien sugirió que existían ciertos patrones del diseño de edificios que eran comunes e inherentemente interesantes y efectivos. El patrón es una descripción del problema y la esencia de su solución, de forma que la solución se pueda reutilizar en diferentes situaciones. (Sommerville, 2005)

El presente proyecto utilizará el concepto de patrón de diseño para especificar la parte más baja de la arquitectura planteada, la capa de datos y la capa de acceso a datos (especificada en el modelo de datos), ambas descritas más adelante en el esquema general de la solución a implementar. Para ello se utilizará el Patrón denominado Patrón de Repositorio (The Repository Pattern), especificado por la Microsoft en su Guía de N-Capas, para definir los objetos que se utilizarán como mediadores entre la capa de negocio y la capa de datos.

"Repository" es una de las formas bien documentadas de trabajar con una fuente de datos. Otra vez Martin Fowler en su libro *Patterns of Application Architecture* describe un repositorio de la siguiente forma:

"Un repositorio realiza las tareas de intermediario entre las capas de modelo de dominio y mapeo de datos, actuando de forma similar a una colección en memoria de objetos del dominio. Los objetos cliente construyen de forma declarativa consultas y las envían a los repositorios para que las satisfagan. Conceptualmente, un repositorio encapsula a un conjunto de objetos almacenados en la base de datos y las operaciones que sobre ellos pueden realizarse, proveyendo de una forma más cercana a la orientación a objetos de la vista de la capa de persistencia. Los repositorios, también soportan el objetivo de separar claramente y en una dirección la dependencia entre el dominio de trabajo y el mapeo o asignación de los datos".

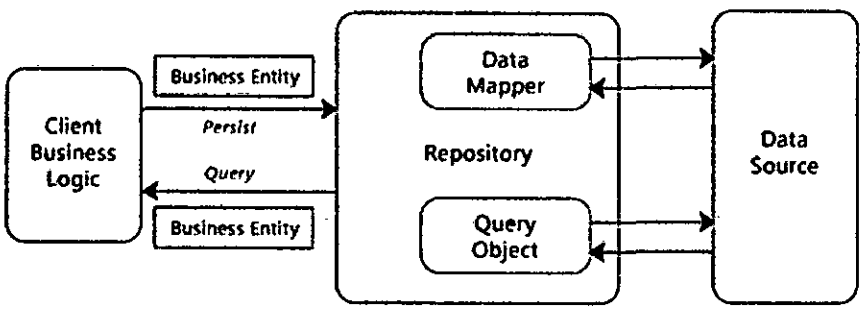


Figura 2.2.2. El Patrón Repositorio
Fuente: MSDN Microsoft - The Repository Pattern

Reutilización basada en generadores

El concepto de reutilización mediante patrones tiene que ver con la descripción del concepto de una forma abstracta y dejando al desarrollador la labor de crear una implementación. Una aproximación alternativa a ésta es la reutilización basada en generadores (Biggerstaff, 1998).

Esta aproximación es un concepto clave en la definición de la solución a brindar por el presente proyecto ya que aquí entran a tallar las mejoras que se van a implementar en la arquitectura de N-Capas utilizando tecnología de Lenguaje de Dominio Específico (DSL por las siglas en Inglés de Domain Specific Language) para la generación de código que permita la implementación del Patrón de Repositorio.

En esta aproximación, el conocimiento reutilizable se captura en un sistema generador de programas que puede ser programado por expertos en el dominio utilizando un lenguaje orientado a dominios o una herramienta CASE interactiva que soporte la generación de sistemas. La descripción de la aplicación específica, de forma abstracta, qué componentes reutilizables tienen que usarse y cómo tienen que ser combinados y parametrizados. Utilizando esta información se puede generar un sistema software operativo. (Sommerville, 2005)

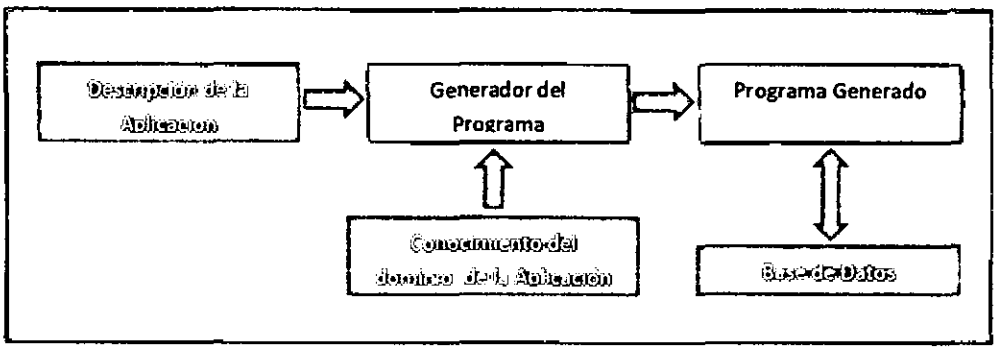


Figura 2.2.3. Reutilización basada en generadores
Fuente: Ingeniería del Software (Sommerville, 2005)

La reutilización basada en generadores ha tenido éxito sobre todo para sistemas de aplicaciones de negocios, y existen muchos productos generadores de aplicaciones de negocios disponibles. Estos pueden generar aplicaciones completas o pueden automatizar parcialmente la creación de aplicaciones y dejar que el programador las complete con detalles específicos. La aproximación a la reutilización basada en generadores también se utiliza en otras áreas, incluyendo:

Los expertos del dominio utilizan un lenguaje específico del dominio para componer estos componentes y crear aplicaciones. Sin embargo, existe un alto coste inicial en la definición e implementación de los conceptos del dominio y en el lenguaje de composición. Esto significa que muchas compañías son reacias a asumir los riesgos de adoptar esta aproximación. (Sommerville, 2005)

Lenguaje específico del dominio (DSL)

Cómo se mencionó anteriormente el sistema generador de programas al que hace alusión Sommerville se va a implementar utilizando un lenguaje específico de dominio brindado por Microsoft como producto de extensibilidad de soluciones basadas en su plataforma .Net.

La Microsoft en su foro de desarrolladores MSDN (Microsoft Developer Network) documenta el DSL de la siguiente manera:

> A diferencia de un lenguaje de uso general como C# o UML, un lenguaje específico (DSL) está diseñado para expresar instrucciones en un espacio del problema, o el dominio.
>
> DSLs bien conocidas incluyen expresiones regulares y SQL. Cada DSL es mucho mejor que un lenguaje de propósito general para describir operaciones en cadenas de texto o una base de datos, pero mucho peor para describir las ideas que están fuera de su propio ámbito. Las industrias individuales tienen sus propias DSLs. Por ejemplo, en la industria de las telecomunicaciones, los idiomas descriptivos de llamada son ampliamente utilizados para especificar la secuencia de estados en una llamada telefónica, y en la industria de viajes una norma DSL se utiliza para describir reservas del vuelo.

Lo opuesto es:

Un lenguaje de programación de propósito general, como por ejemplo C o Java o un lenguaje de modelaje de propósito general como UML.

Crear un lenguaje específico del dominio (con software que lo soporte) vale la pena cuando permite que un tipo particular de problemas o soluciones puedan ser expresadas más claramente que con otros lenguajes existentes, y el tipo de problema en cuestión reaparece lo suficiente. La programación orientada a lenguajes considera la creación de lenguajes específicos para expresar problemas una parte estándar para el proceso de solucionar el problema.

En los DSL, la semántica del lenguaje está muy cercana al dominio del problema para el cual se diseña. Tienen un alto nivel de abstracción al usuario, por tanto, están dirigidos a expertos en el dominio.

Esta tecnología permite modelar de una forma gráfica los componentes de un dominio específico y permiten generar código, para este caso permitirán modelar los objetos de acceso a datos y lógica del negocio del framework a desarrollar.

Entity Framework es un claro ejemplo del uso de DSL por parte de la Microsoft para modelar el repositorio de datos.

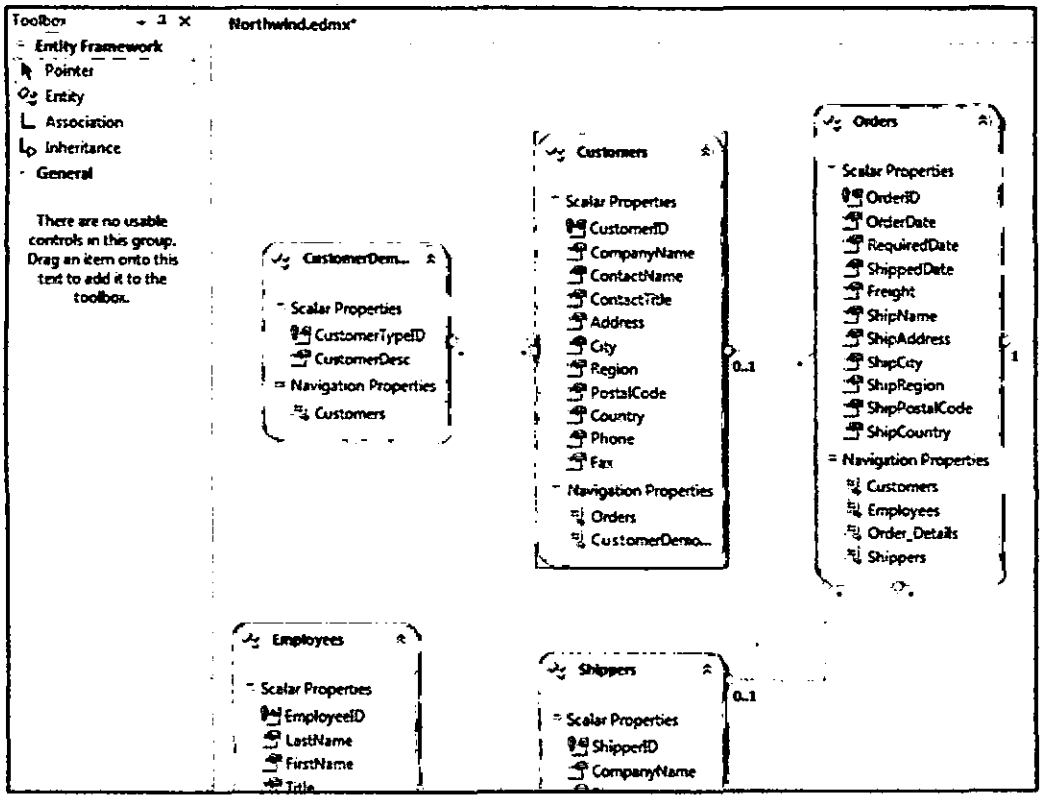


Figura 2.2.4. Modelado de datos mediante Entity Framework
Fuente: MSDN Microsoft – Entity Framework

Este mecanismo de generación de código define como dominio el repositorio de datos y las clases de acceso a datos utilizando ADO .Net para el acceso a los diferentes orígenes de datos y que sirve como mediador entre el aplicativo y el origen de datos mismo como muestra el siguiente gráfico del dominio:

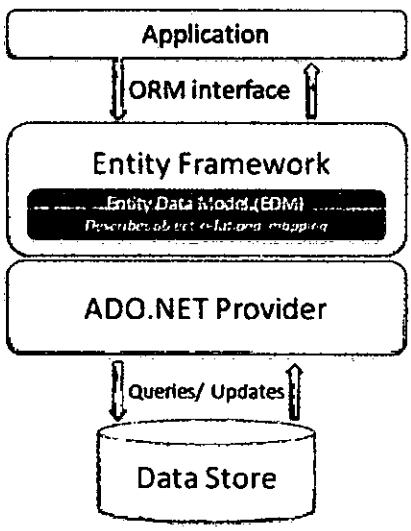


Figura 2.2.5. Arquitectura Entity Framework
Fuente: MSDN Microsoft – Entity Framework

Ventajas de desarrollo específico de dominio

Un lenguaje específico puede proporcionar las siguientes ventajas:

- Contiene las construcciones que se ajuste a exactamente el espacio del problema.
  A diferencia de lenguajes de uso general, un lenguaje específico consta de los elementos y las relaciones que representan directamente la lógica del espacio del problema. Por ejemplo, una aplicación de la póliza de seguros debe incluir los elementos para las directivas y las peticiones. Un lenguaje específico facilita diseñar la aplicación, y encuentra y corregir errores de lógica.

- Permite a los no desarrolladores de software y las personas que no conocen el dominio entienden el diseño global.
  Mediante un lenguaje específico del gráfico, puede crear una representación visual del dominio de modo que los no desarrolladores de software pueden fácilmente entender el diseño de la aplicación.

- Facilita la creación un prototipo de la aplicación final.
  Los desarrolladores pueden utilizar el código que el modelo genera para crear una aplicación de prototipo que pueden mostrar a los clientes.

Finalmente, haciendo uso de las dos aproximaciones de reutilización anteriores se implementará un marco de trabajo, que denominaremos Framework de N-Capas de ahora en adelante, para definir la arquitectura del proyecto de software sobre el cual se realizará el estudio del presente proyecto. A continuación se presenta una breve definición teórica de los marcos de trabajo.

Marcos de trabajo de aplicaciones (FRAMEWORKS DE DESARROLLO)

Definición de Marco de Trabajo

Un marco de trabajo (o marco de trabajo de aplicaciones) es un diseño de un subsistema formado por una colección de clases concretas y abstractas y la interfaz entre ellas (Wirfs-Brock y Johnson, 1990). Los detalles particulares del subsistema de aplicación son implementados añadiendo componentes y proporcionando implementaciones concretas de clases abstractas en el marco de trabajo. Los marcos de trabajo raramente son aplicaciones por sí mismos. Las aplicaciones se construyen normalmente integrando varios marcos conceptuales de trabajo.

Fayad y Schmidt (Fayad y Schmidt, 1997) describen tres clases de marcos de trabajo:

1. **Marcos de trabajo de infraestructura de sistemas.** Estos marcos de trabajo soportan el desarrollo de infraestructuras de sistemas tales como comunicaciones, interfaces de usuario y compiladores (Schmidt, 1997).
2. **Marcos de trabajo para la integración de middleware.** Consisten en un conjunto d estándares y clases de objetos asociados que soportan la comunicación de componentes y el intercambio de información. Ejemplos de este tipo de marcos son COFBA, COM+ de Microsoft y Enterprise Java Beans.
3. **Marcos de trabajo de aplicaciones empresariales.** Se refieren a dominios de aplicaciones específicos tales como telecomunicaciones o sistemas financieros (Baumer et al, 1997). Estos marcos de trabajo encapsulan el conocimiento del dominio de la aplicación y soportan el desarrollo de aplicaciones para los usuarios finales.

El framework resultante de la Guía Microsoft supone un Framework o Marco de Trabajo de Aplicaciones Empresariales, sobre el cual se va a basar el desarrollo del proyecto de investigación para aplicar la propuesta de solución.

2.3. Propuesta de Solución

Framework de desarrollo N-Capas Orientado al Domino

Orientación a tendencias de Arquitectura DDD (Domain Driven Design)

El objetivo de esta arquitectura marco es proporcionar una base consolidada y guías de arquitectura para un tipo concreto de aplicaciones: Aplicaciones empresariales complejas. Este tipo de aplicaciones se caracterizan por tener una vida relativamente larga y un volumen de cambios evolutivos considerable. Por lo tanto, en estas aplicaciones es muy importante todo lo relativo al mantenimiento de la aplicación, la facilidad de actualización, o la sustitución de tecnologías y frameworks/ORMs (Objectrelational mapping) por otras versiones más modernas o incluso por otros diferentes, etc. El objetivo es que todo esto se pueda realizar con el menor impacto posible sobre el resto de la aplicación. En definitiva, que los cambios de tecnologías de infraestructura de una aplicación no afecten a capas de alto nivel de la aplicación, especialmente, que afecten lo mínimo posible a la capa del Dominio de la aplicación. (De La Torre Llorente, 2010)

Capas de Presentación, Aplicación, Dominio e Infraestructura

En el nivel más alto y abstracto, la vista de arquitectura lógica de un sistema puede considerarse como un conjunto de servicios relacionados agrupados en diversas capas, similar al siguiente esquema (siguiendo las tendencias de Arquitectura DDD):

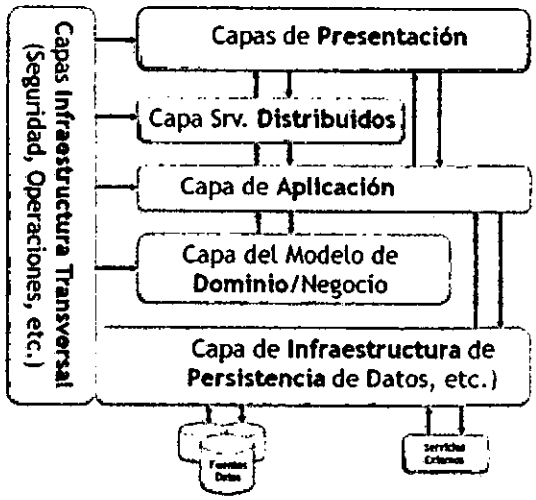


Figura 2.3.1. Vista de arquitectura lógica simplificada de un sistema N-Capas DDD
Fuente: Guía de Arquitectura de N-Capas Orientado al Dominio (De La Torre Llorente, 2010)

Este es un modelo a nivel Macro de la Arquitectura de N-Capas orientada al Dominio que se va a emplear para a generación de la arquitectura del proyecto.

Arquitectura marco N-Capas con Orientación al Dominio

El objetivo de esta arquitectura marco es estructurar de una forma limpia y clara la complejidad de una aplicación empresarial basada en las diferentes capas de la arquitectura, siguiendo el patrón N-Layer y las tendencias de arquitecturas en DDD.
El patrón N-Layer distingue diferentes capas y sub-capas internas en una aplicación, delimitando la situación de los diferentes componentes por su tipología.

Por supuesto, esta arquitectura concreta N-Layer es personalizable según las necesidades de cada proyecto y/o preferencias de Arquitectura. Simplemente proponemos una Arquitectura marco a seguir que sirva como punto base a ser modificada o adaptada por arquitectos según sus necesidades y requisitos.

En concreto, las capas y sub-capas propuestas para aplicaciones N-Layered con Orientación al Dominio son:

Arquitectura N-Capas con Orientación al Dominio

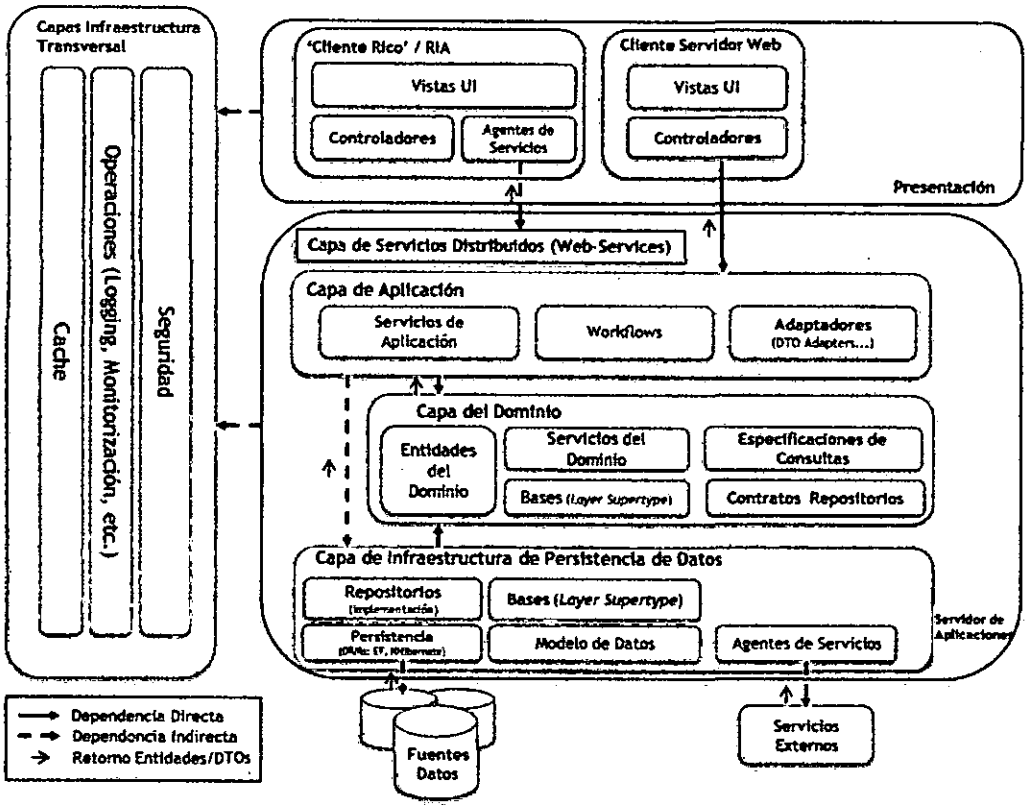


Figura 2.3.2. Arquitectura N-Capas con Orientación al Dominio
Fuente: Guía de Arquitectura de N-Capas Orientado al Dominio (De La Torre Llorente, 2010)

A continuación una breve descripción de cada una de las capas y la lista de cada uno de los componentes por capa de esta arquitectura:

Capa de Presentación
Esta capa es responsable de mostrar información al usuario e interpretar sus acciones. Los componentes de las capas de presentación implementan la funcionalidad requerida para que los usuarios interactúen con la aplicación.

- Subcapas de Componentes Visuales (Vistas o Formularios)
- Subcapas de Proceso de Interfaz de Usuario (Controladores y similares)

Capa de Servicios Distribuidos (Servicios-Web)
Cuando una aplicación actúa como proveedor de servicios para otras aplicaciones remotas, o incluso si la capa de presentación esta también localizada fisicamente en localizaciones remotas (aplicaciones Rich-Client, RIA, OBA, etc.), normalmente se publica la lógica de negocio (capas de negocio internas) mediante una capa de servicios. Esta capa de servicios (habitualmente Servicios Web) proporciona un medio de acceso remoto basado en canales de comunicación y mensajes de datos. Es importante destacar que esta capa debe ser lo más ligera posible y que no debe incluir nunca 'lógica' de negocio. Hoy por hoy, con las tecnologías actuales hay muchos elementos de una arquitectura que son muy simples de realizar en esta capa y en muchas ocasiones se tiende a incluir en ella propósitos que no le competen.

- Servicios-Web publicando las Capas de Aplicación y Dominio

Capa de Aplicación (Lógica de Negocio)
Esta capa forma parte de la propuesta de arquitecturas orientadas al Dominio. Define los trabajos que la aplicación como tal debe de realizar y redirige a los objetos del dominio y de infraestructura (persistencia, etc.) que son los que internamente deben resolver los problemas.

- Servicios de Aplicación (Tareas y coordinadores de casos de uso)
- Adaptadores (Conversores de formatos, etc.)
- Subcapa de Workflows (Opcional)
- Clases base de Capa Aplicación (Patrón Layer-Supertype)

Capa del Modelo de Dominio
Esta capa es responsable de representar conceptos de negocio, información sobre la situación de los procesos de negocio e implementación de las reglas del dominio.
También debe contener los estados que reflejan la situación de los procesos de negocio.
Esta capa Dominio es el corazón del software. Normalmente podemos definir los siguientes elementos dentro de la capa de Dominio:

- Entidades del Dominio
- Servicios del Dominio
- Especificaciones de Consultas (Opcional)
- Contratos/Interfaces de Repositorios
- Clases base del Dominio (Patrón Layer-Supertype)

Capa de Infraestructura de Acceso a Datos
Esta capa proporciona la capacidad de persistir datos así como lógicamente acceder a ellos. Pueden ser datos propios del sistema o incluso acceder a datos expuestos por sistemas externos (Servicios Web externos, etc.). Así pues, esta capa de persistencia de datos expone el acceso a datos a las capas superiores, normalmente las capas del dominio. Esta exposición deberá realizarse de una forma desacoplada.

- Implementación de Repositorios
- Modelo lógico de Datos
- Clases Base (Patrón Layer-Supertype)
- Infraestructura tecnología ORM
- Agentes de Servicios externos

Componentes/Aspectos Horizontales de la Arquitectura
Proporcionan capacidades técnicas genéricas que dan soporte a capas superiores. En definitiva, son „bloques de construcción" ligados a una tecnología concreta para desempeñar sus funciones.

- Aspectos horizontales de Seguridad, Gestión de operaciones,
- Monitorización, Correo Electrónico automatizado, etc.

De esta lista de componentes se van a aplicar las mejoras en la propuesta de solución empleando la generación de código mediante DSL para los siguientes componentes.

Capa de Aplicación (Lógica de Negocio)
- Servicios de Aplicación (Tareas y coordinadores de casos de uso)
- Adaptadores (Conversores de formatos, etc.)

Capa del Modelo de Dominio
- Entidades del Dominio
- Contratos/Interfaces de Repositorios
- Clases base del Dominio (Patrón Layer-Supertype)

Capa de Infraestructura de Acceso a Datos
- Implementación de Repositorios
- Modelo Lógico de Datos
- Clases Base (Patrón Layer-Supertype)

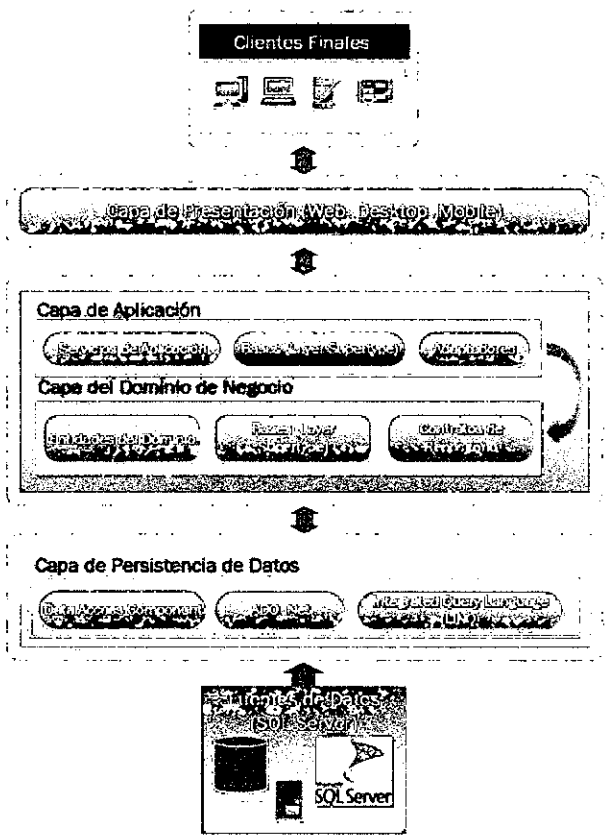


Figura 2.3.3. Componentes de la Arquitectura sobre los cuales se va a aplicar DSL
Fuente: Elaboración Propia

El esquema anterior muestra el detalle de los componentes a generar, obviando la capa de presentación ya que es independiente al propósito final, es decir el alcance del frameworks se centrará en la lista expuesta líneas arriba que incluye la capa de datos, capa de acceso a datos y la capa de negocio. Para ello se implementará un marco de trabajo referencial con esta arquitectura definida por la Guía de N-Capas.

Finalmente como conclusión para el marco teórico se citan a continuación las ventajas y desventajas del uso de esta arquitectura de acuerdo a la Guía Microsoft.

Ventajas del uso de Arquitecturas N-Capas

- ✓ Desarrollo estructurado, homogéneo y similar de las diferentes aplicaciones de una organización.
- ✓ Facilidad de mantenimiento de las aplicaciones pues los diferentes tipos de tareas están siempre situados en las mismas áreas de la arquitectura.
- ✓ Fácil cambio de tipología en el despliegue físico de una aplicación (2-Tier, 3-Tier, etc.), pues las diferentes capas pueden separarse físicamente de forma fácil.

Desventajas del uso de Arquitecturas N-Capas

- ✓ En el caso de aplicaciones muy pequeñas, estamos añadiendo una complejidad excesiva (capas, desacoplamiento, etc.). Pero este caso es muy poco probable en aplicaciones empresariales con cierto nivel.

CAPÍTULO III: METODOLOGÍA: MÉTODOS Y MATERIALES

3.1. Tipo de Investigación

Según el objeto de estudio, la presente investigación es de tipo causal porque de acuerdo a (Namakforoosh, 2005) se espera que una variable independiente produzca ciertos cambios en la variable dependiente.

Según el tratamiento de que se da al objeto de estudio, es una investigación causal ya que se realizará la causa efecto de la ausencia o presencia del framework de N-Capas orientado al dominio para el desarrollo de un aplicativo empresarial.

3.2. El Modelo Teórico

Se ha definido el presente trabajo de investigación cómo un prototipo de desarrollo evolutivo ya que define una serie de reglas definidas en un framework de desarrollo de software basado en un prototipo tal y como se describe en el marco teórico.

3.3. Diseño de la Investigación

El presente trabajo de investigación es de tipo no experimental y se centra en el desarrollo de mejoras aplicadas al framework de N-Capas orientado al dominio propuesto por Microsoft, estas mejoras están basadas en tecnología DSL de meta modelado y generación de código. La investigación consta de varias etapas las cuales se describen a continuación.

1. Recopilar Información
En esta etapa de la investigación se recogerá información técnica acerca de las tecnologías a utilizar y se definirá el entorno de desarrollo sobre el cual se va a trabajar durante el resto del proyecto.

Recopilación de Información sobre la plataforma base
- ✓ Microsoft .Net Platform.
- ✓ SQL Server 2012.
- ✓ Domain Specific Language (DSL).

Recopilación de Información sobre el software a utilizar
- ✓ Microsoft Visual Studio 2012.
- ✓ Microsoft Visual Studio Extensibility.
- ✓ Microsoft Visual Studio Online.

2. Identificar Requerimientos de Mejoras
En esta etapa de la investigación se definirán los componentes de la arquitectura de la solución a implementar sobre los cuales se aplicará la tecnología DSL, para ello se utilizará como base la guía de Microsoft de la arquitectura de N-Capas orientada al dominio y se tomará en cuenta el alcance del proyecto .

3. Diseñar la Solución Propuesta
Definidos los componentes sobre los cuales se van a implementar las mejoras se diseñará la arquitectura final de la solución a implementar basada en las tecnologías definidas en el punto 1) del diseño de la investigación.

4. Desarrollar la Solución Propuesta
En esta etapa del proceso de investigación se realizará el desarrollo de la solución propuesta utilizando la tecnología y la arquitectura definidas en los puntos anteriores.

5. Probar la Solución Implementada
Cuando se tenga construida la solución propuesta, en esta etapa del estudio se realizarán las pruebas de la misma para estudiar el comportamiento de las variables y definir la hipótesis de investigación.

6. Documentar la Solución Implementada
En el paso final de la investigación se realizará la siguiente documentación sobre la solución propuesta.

- Documento de Arquitectura
- Manual de Usuario

3.4. Métodos e Instrumentos Cuantitativos y/o Cualitativos de Medición

SCRUM

La metodología SCRUM se utilizó mediante la herramienta Visual Studio Online, una herramienta gratuita de Microsoft para un máximo de 5 miembros por equipo, este portal web posee una plantilla Scrum 3.0 para llevar el proceso de desarrollo de software y provee también un repositorio de código fuente para llevar el control de versiones de las fuentes de la solución a implementar en el presente proyecto.

Roles Scrum: Se han definido los siguientes roles para el proyecto:

- ✓ **Product Owner:** Ing. Arturo Sandoval Rivera (Asesor)
- ✓ **Scrum Manager:** Ing. Arturo Sandoval Rivera (Asesor)
- ✓ **Scrum Team:** Ing. Nelsson José Aguilar Salvador (Tesista)

Iteraciones: Se han definido las siguientes iteraciones Scrum para el proyecto

Release 1: Se ha definido un solo Release para el proyecto, este Release a su vez tiene un solo Sprint:

Sprint 1:	Fecha de Inicio:	16/03/2015
	Fecha de Fin:	30/04/2015

Esfuerzo: El esfuerzo por cada miembro del equipo es el siguiente:

- ✓ **Scrum Team:** Ing. Arturo Sandoval Rivera (Asesor)
  (1 horas diarias x 6 días = 6 horas)
  Nelsson José Aguilar Salvador (Tesista)
  (6 horas diarias x 40 días = 240 horas)

Se consideran los días de semana de lunes a sábado para el tesista y los domingos para la revisión por parte del asesor.

Product Backlog: Se han definido los siguientes PBIs para el Product Backlog el proyecto de tesis:

ID	PBI	Estado	Actividad	Tipo
118	Especificación del Planteamiento del Problema	Committed		PBI
142	Especificación del Planteamiento del Problema	Done	Documentation	Task
119	Especificación de la Importancia y Justificación	Committed		PBI
143	Especificación de la Importancia y Justificación	Done	Documentation	Task
120	Especificación de los Objetivos de la Investigación	Committed		PBI
144	Especificación de los Objetivos de la Investigación	Done	Documentation	Task
121	Elaboración del Marco Teórico del Proyecto	Committed		PBI
145	Elaboración del Marco Teórico del Proyecto	Done	Documentation	Task
122	Especificación de las Variables de la Investigación y Correlación	Committed		PBI
146	Especificación de las Variables de la Investigación y Correlación	Done	Documentation	Task
123	Especificación del Diseño de la Investigación	Committed		PBI
147	Especificación del Diseño de la Investigación	Done	Documentation	Task
124	Especificación del Cronograma de Actividades y Presupuesto	Committed		PBI
148	Especificación del Cronograma de Actividades y Presupuesto	Done	Documentation	Task
125	Especificación de la plataforma a utilizar	Committed		PBI
149	Especificación de la plataforma a utilizar	Done	Design	Task
126	Especificación de la tecnología a utilizar	Committed		PBI
150	Especificación de la tecnología a utilizar	Done	Design	Task
127	Identificación de los componentes de desarrollo iterativos	Committed		PBI
151	Identificación de los componentes de desarrollo iterativos	Done	Requirements	Task
128	Identificación de la Arquitectura del Proyecto	Committed		PBI
152	Identificación de la Arquitectura del Proyecto	Done	Requirements	Task
129	Diseño de los Componentes de la Arquitectura de la Solución	Committed		PBI
156	Diseño de la Capa de Entidad	Done	Design	Task
157	Diseño de la Capa de Acceso a Datos	Done	Design	Task
158	Diseño de la Capa de Datos	Done	Design	Task
116	Diseño de la Arquitectura de la Solución	Committed		PBI
155	Diseño de la Estructura del Proyecto	Done	Design	Task
131	Desarrollo del Componente de Entidad	Committed		PBI
160	Desarrollo del Componente de Entidad	Done	Development	Task
132	Desarrollo del Componente de Base de Datos	Committed		PBI
161	Desarrollo del Componente de Base de Datos	Done	Development	Task
133	Desarrollo del Componente de Acceso a Datos	Committed		PBI
162	Desarrollo del Componente de Acceso a Datos	Done	Development	Task
130	Desarrollo de la Plantilla de Arquitectura	Committed		PBI
159	Desarrollo de la Plantilla de Arquitectura	Done	Development	Task
134	Prueba de Acoplamiento de la Arquitectura al framework N-Capas	Committed		PBI
163	Prueba de Acoplamiento de la Arquitectura al framework N-Capas	Done	Testing	Task
135	Prueba del Componente de Entidad	Committed		PBI
164	Prueba del Componente de Entidad	Done	Testing	Task

136	Prueba del Componente de Modelo de Datos	Committed		PBI
165	Prueba del Componente de Modelo de Datos	Done	Testing	Task
137	Prueba del Componente de Acceso a Datos	Committed		PBI
166	Prueba del Componente de Acceso a Datos	Done	Testing	Task
138	Elaboración del Documento de Arquitectura	Committed		PBI
167	Elaboración del Documento de Arquitectura	Done	Documentation	Task
139	Elaboración del Manual de Usuario	Committed		PBI
168	Elaboración del Manual de Usuario	Done	Documentation	Task
140	Elaboración del Informe Final	Committed		PBI
169	Elaboración del Informe Final	Done	Documentation	Task
141	Sustentación del Proyecto	Committed		PBI
170	Sustentación del Proyecto	Done	Deployment	Task

Tabla 3.4.1. Product Backlog del Proyecto
Fuente: Ingeniería del Software (Sommerville, 2005)

Proceso de avance de Proyecto (Burndown Chart): El siguiente gráfico muestra el proceso de avance del proyecto del Sprint 1.

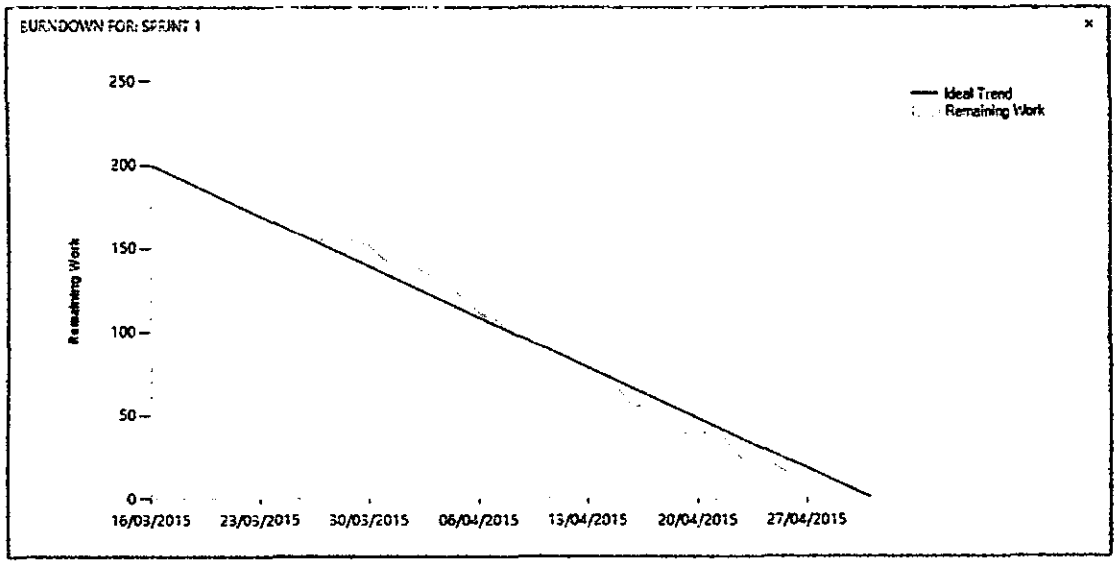


Figura 3.4.1. Proceso de avance del proyecto
Fuente: Elaboración Propia

Dirección Web del Proyecto: Se puede ingresar el proyecto de Visual Studio en el siguiente enlace:

https://devframework.visualstudio.com/DefaultCollection/DevFramework

Visual Studio 2012 Test Manager

Esta herramienta se utilizó para las pruebas de integración y pruebas unitarias sobre los elemento del diseño de la arquitectura de aplicación diseñada para la solución, esta herramienta de pruebas forma parte del pack de Visual Studio 2012.

Se aplicará pruebas de integración a:

- ✓ Arquitectura del Proyecto

Se aplicará pruebas unitarias a los siguientes elementos de la arquitectura.

- ✓ Elemento de Acceso a Datos

Proyecto de Pruebas: Se ha creado un proyecto de pruebas en base al proyecto de Visual Studio Online.

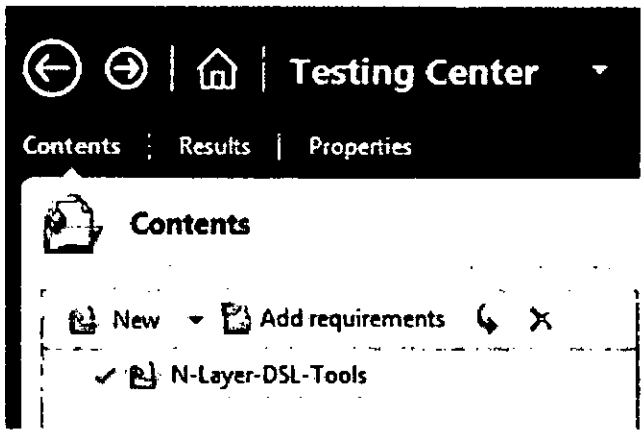
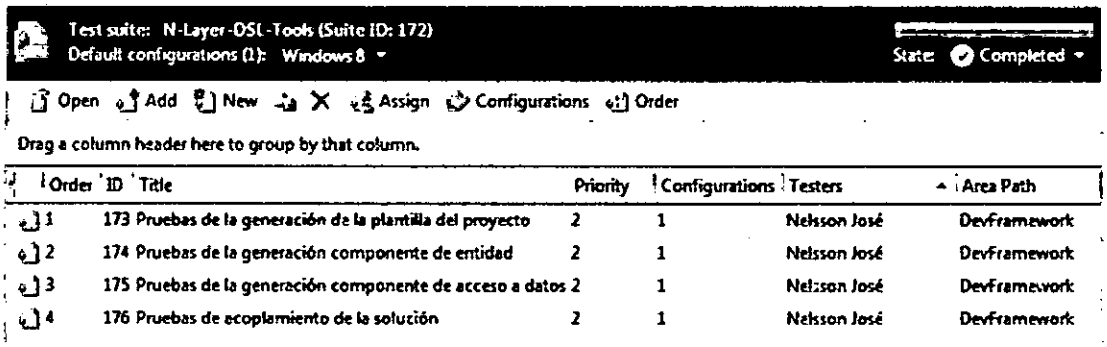


Figura 3.4.2. Proyecto de pruebas de la solución
Fuente: Elaboración Propia

Set de Pruebas: Las pruebas definidas para el proyecto son las siguientes:



Order	ID	Title	Priority	Configurations	Testers	Area Path
1	173	Pruebas de la generación de la plantilla del proyecto	2	1	Nelsson José	Devframework
2	174	Pruebas de la generación componente de entidad	2	1	Nelsson José	Devframework
3	175	Pruebas de la generación componente de acceso a datos	2	1	Nelsson José	Devframework
4	176	Pruebas de acoplamiento de la solución	2	1	Nelsson José	Devframework

Figura 3.4.3. Set de pruebas
Fuente: Elaboración Propia

CAPÍTULO IV: RECOPILACIÓN DE LA INFORMACIÓN

4.1. Recopilación de Información sobre la plataforma base

4.1.1. Microsoft .Net Platform

.NET Framework es un componente integral de Windows que admite la compilación y la ejecución de la siguiente generación de aplicaciones y servicios Web XML. El diseño de .NET Framework está enfocado a cumplir los siguientes objetivos:

- ✓ Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.

- ✓ Proporcionar un entorno de ejecución de código que minimiza los conflictos en el despliegue y versionado de software.

- ✓ Ofrecer un entorno de ejecución de código que promueva la ejecución segura del mismo, incluso del creado por terceros desconocidos o que no son de plena confianza.

- ✓ Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan scripts o intérpretes de comandos.

- ✓ Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.

- ✓ Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código.

.NET Framework contiene dos componentes principales: Common Language Runtime y la Biblioteca de clases de .NET Framework.

Common Language Runtime es el fundamento de .NET Framework. El motor en tiempo de ejecución se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la comunicación remota, al tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que promueven su seguridad y solidez. De hecho, el concepto de administración de código es un principio fundamental del motor en tiempo de ejecución. El código destinado al motor en tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado. La biblioteca de clases, el otro componente principal de .NET Framework, es una completa colección orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones que abarcan desde las tradicionales herramientas de interfaz gráfica de usuario (GUI) o de línea de comandos hasta las aplicaciones basadas en las innovaciones más recientes proporcionadas por ASP.NET, como los formularios Web Forms y los servicios Web XML. ("Información general y conceptual sobre .NET Framework", 2015)

.NET Framework puede hospedarse en componentes no administrados que cargan Common Language Runtime en sus procesos e inician la ejecución de código administrado, con lo que se crea un entorno de software en el que se pueden utilizar características administradas y no administradas. En .NET Framework no sólo se ofrecen varios hosts de motor en tiempo de ejecución, sino que también se admite el desarrollo de estos hosts por parte de terceros.

Por ejemplo, ASP.NET hospeda el motor en tiempo de ejecución para proporcionar un entorno de servidor escalable para el código administrado. ASP.NET trabaja directamente con el motor en tiempo de ejecución para habilitar aplicaciones de ASP.NET y servicios Web XML, que se tratan más adelante en este tema.

Internet Explorer es un ejemplo de aplicación no administrada que hospeda el motor en tiempo de ejecución (en forma de una extensión de tipo MIME). Al usar Internet Explorer para hospedar el motor en tiempo de ejecución, puede incrustar componentes administrados o controles de Windows Forms en documentos HTML. Al hospedar el motor en tiempo de ejecución de esta manera se hace posible el uso de código móvil administrado (similar a los controles de Microsoft® ActiveX®), pero con mejoras significativas que sólo el código administrado puede ofrecer, como la ejecución con confianza parcial y el almacenamiento aislado de archivos.

En la ilustración siguiente se muestra la relación de Common Language Runtime y la biblioteca de clases con las aplicaciones y el sistema en su conjunto. En la ilustración se representa igualmente cómo funciona el código administrado dentro de una arquitectura mayor. ("Información general y conceptual sobre .NET Framework", 2015)

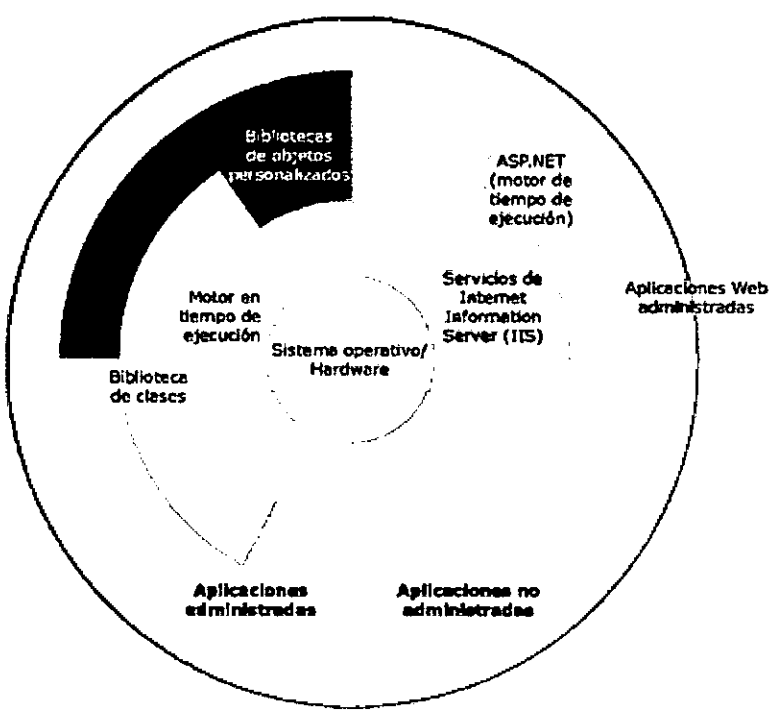


Figura 4.1.1. .NET Framework en contexto
Fuente: MSDN Microsoft – Información general y conceptual sobre .NET Framework

La biblioteca de clases de .NET Framework es una colección de tipos reutilizables que se integran estrechamente con Common Language Runtime. La biblioteca de clases está orientada a objetos, lo que proporciona tipos de los que su propio código administrado puede derivar funciones. Esto ocasiona que los tipos de .NET Framework sean sencillos de utilizar y reduce el tiempo asociado con el aprendizaje de las nuevas características de .NET Framework. Además, los componentes de terceros se pueden integrar sin dificultades con las clases de .NET Framework.

Por ejemplo, las clases de colección de .NET Framework implementan un conjunto de interfaces que puede usar para desarrollar sus propias clases de colección. Éstas se combinarán fácilmente con las clases de .NET Framework. ("Información general y conceptual sobre .NET Framework", 2015)

4.1.2. SQL Server 2012

Microsoft® SQL Server™ es un sistema de administración y análisis de bases de datos relacionales de Microsoft para soluciones de comercio electrónico, línea de negocio y almacenamiento de datos. En esta sección, encontrará información sobre varias versiones de SQL Server. También encontrará artículos sobre bases de datos y aplicaciones de diseño de bases de datos así como ejemplos de los usos de SQL Server.

Microsoft SQL Server 2012 se basa en las funciones críticas ofrecidas en la versión anterior, proporcionando un rendimiento, una disponibilidad y una facilidad de uso innovadores para las aplicaciones más importantes. Microsoft SQL Server 2012 ofrece nuevas capacidades en memoria en la base de datos principal para el procesamiento de transacciones en línea (OLTP) y el almacenamiento de datos, que complementan nuestras capacidades de almacenamiento de datos en memoria y BI existentes para lograr la solución de base de datos en memoria más completa del mercado.

SQL Server 2012 también proporciona nuevas soluciones de copia de seguridad y de recuperación ante desastres, así como de arquitectura híbrida con Windows Azure, lo que permite a los clientes utilizar sus actuales conocimientos con características locales que aprovechan los centros de datos globales de Microsoft. Además, SQL Server 2012 aprovecha las nuevas capacidades de Windows Server 2012 y Windows Server 2012 R2 para ofrecer una escalabilidad sin parangón a las aplicaciones de base de datos en un entorno físico o virtual. ("Microsoft SQL Server", 2015)

4.1.3. Domain Specific Language (DSL)

A diferencia de un lenguaje de uso general como C# o UML, un lenguaje específico (DSL) está diseñado para expresar instrucciones en un espacio del problema, o el dominio.

DSLs bien conocidas incluyen expresiones regulares y SQL. Cada DSL es mucho mejor que un lenguaje de propósito general para describir operaciones en cadenas de texto o una base de datos, pero mucho peor para describir las ideas que están fuera de su propio ámbito. Las industrias individuales tienen sus propias DSLs. Por ejemplo, en la industria de las telecomunicaciones, los idiomas descriptivos de llamada son ampliamente utilizados para especificar la secuencia de estados en una llamada telefónica, y en la industria de viajes una norma DSL se utiliza para describir reservas del vuelo.

El negocio y el proyecto también se ocupan de conjuntos especiales de conceptos que se puede describir con DSL. Por ejemplo, podría definir un DSL para una de estas aplicaciones:

- Plan de rutas de navegación de un sitio Web.
- Esquemas eléctricos para componentes electrónicos.
- Redes de las bandas transportadoras y de equipaje que administran el equipo de un aeropuerto.

Cuando diseñe DSL, defina una clase de dominio para cada uno de los conceptos importantes en el dominio, como una página Web, una lámpara, o un mostrador de facturación de aeropuerto. Al definir las relaciones de dominio como hipervínculo, conexión, o una banda transportadora para vincular los conceptos juntos.

Los usuarios del DSL crean modelos. Los modelos son instancias del DSL. Por ejemplo, describen un sitio Web concreto, o la conexión de un dispositivo determinado, o de equipaje controlando el sistema de un aeropuerto determinado.

Los usuarios pueden ver un modelo como diagrama o como Windows form. Los modelos se pueden ver como XML's, que es cómo realmente se almacenan. Cuando se define un DSL, se define cómo las instancias de cada clase de dominio y cada relación aparecen en la pantalla del usuario. DSL normal se muestra como una colección de iconos o de rectángulos conectados mediante flechas. ("Acerca de los lenguajes específicos de dominio", 2015)

4.2. Recopilación de Información sobre el software a utilizar

4.2.1. Microsoft Visual Studio 2012

Visual Studio es un conjunto de herramientas de desarrollo de software y de otras tecnologías basado en componentes para crear aplicaciones eficaces de alto rendimiento. Además, Visual Studio está optimizado para diseño basado en equipos, desarrollo e implementación mediante Visual Studio Online o Team Foundation Server.

4.2.2. Microsoft Visual Studio Extensibility (Visual Studio Software Development Kit)

Muchos desarrolladores utilizan Visual Studio para crear y gestionar sus proyectos de desarrollo de software. Microsoft ha proporcionado varias formas de personalizar y extender Visual Studio para automatizar tareas o agregar funciones. Puede crear extensiones de Visual Studio para su propio uso o para su distribución a otros usuarios. Mediante la creación de complementos, se puede personalizar el IDE de Visual Studio para ayudarle a trabajar de manera más eficiente. Para obtener más información, consulte la extensibilidad y automatización de Visual Studio.

Para ampliar aún más Visual Studio, utilice el SDK Estudio Visual. El SDK de Visual Studio es un conjunto de herramientas y documentación que le pueden ayudar a extender Visual Studio o crear nuevas funciones que se integran en Visual Studio. Usted puede

distribuir sus extensiones a otros usuarios. Las siguientes son algunas de las formas en que se puede extender de Visual Studio:

- ✓ Añadir comandos, ventanas y otras características para el IDE.
- ✓ Extienda el editor de Visual Studio.
- ✓ Habilitar el soporte de un nuevo lenguaje.
- ✓ Extienda la funcionalidad de diseño de datos de fuentes de datos externas.
- ✓ Añade tus propias plantillas de tipo proyecto.
- ✓ Integrar control de origen de encargo.
- ✓ Personaliza el depurador de Visual Studio o crear el tuyo propio.
- ✓ Crear y gestionar sus equipos conjuntos de pruebas.

4.2.3. Microsoft Visual Studio Online

Visual Studio Online, que se basa en las capacidades de Team Foundation Server con servicios adicionales en la nube, es el inicio en línea para los proyectos de desarrollo. Póngalo en marcha en tan solo unos minutos en nuestra infraestructura de nube sin tener que instalar ni configurar ningún servidor. Visual Studio Online se conecta con Visual Studio, Eclipse, Xcode y otros clientes Git para apoyar el desarrollo en diferentes plataformas y lenguajes.

Cada cuenta de Visual Studio Online dispone de cinco usuarios gratuitos. A medida que crezcan su equipo de trabajo y sus necesidades, podrá mezclar y combinar recursos y planes de usuarios para proporcionar a cada usuario lo que necesita. Los suscriptores de Visual Studio con MSDN pueden unirse de manera gratuita a proyectos de cuenta exclusivos de los cinco usuarios gratuitos que se incluyen.

4.3. Recopilación de información de la arquitectura de N-Capas orientada al dominio

El marco de N-Capas orientado al dominio propuesto por Microsoft se basa en la arquitectura marco de desarrollo orientado al dominio (Domain Driven Development).

El objetivo de esta arquitectura marco es proporcionar una base consolidada y guías de arquitectura para un tipo concreto de aplicaciones: "Aplicaciones empresariales complejas". Este tipo de aplicaciones se caracterizan por tener una vida relativamente larga y un volumen de cambios evolutivos considerable. Por lo tanto, en estas aplicaciones es muy importante todo lo relativo al mantenimiento de la aplicación, la facilidad de actualización, o la sustitución de tecnologías y frameworks/ORMs (Objectrelational mapping) por otras versiones más modernas o incluso por otros diferentes, etc. El objetivo es que todo esto se pueda realizar con el menor impacto posible sobre el resto de la aplicación. En definitiva, que los cambios de tecnologías de infraestructura de una aplicación no afecten a capas de alto nivel de la aplicación, especialmente, que afecten lo mínimo posible a la capa del "Dominio de la aplicación". (De La Torre Llorente, 2010)

El esquema a nivel macro de la arquitectura de N-Capas orientada al dominio se muestra en el siguiente diagrama:

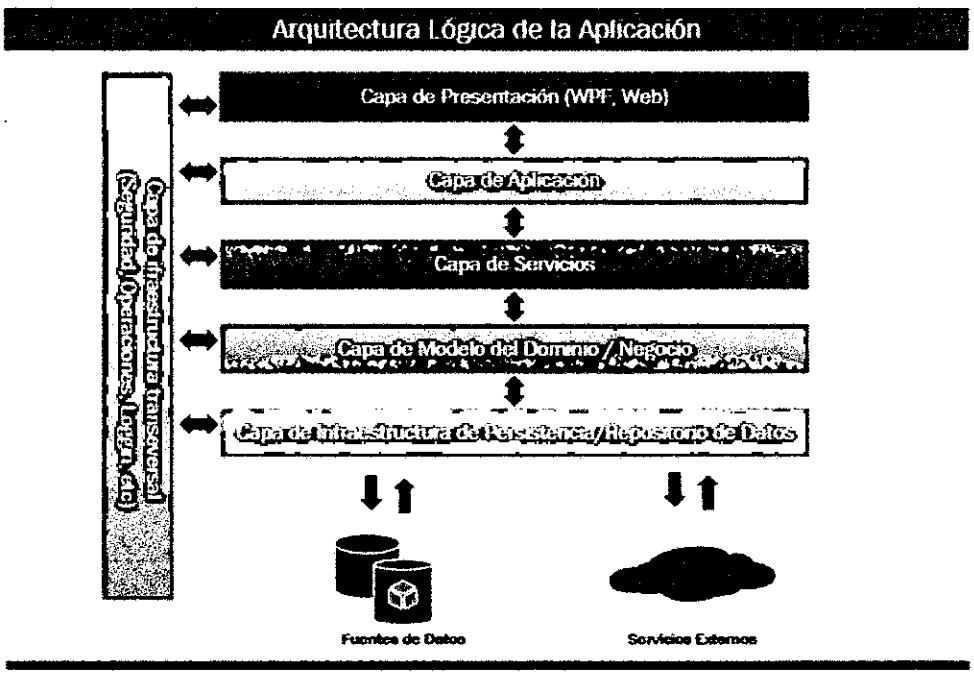


Figura 4.3.1. Arquitectura Lógica de Aplicación
Fuente: Elaboración Propia

El diseño de cada capa será cohesivo, pero delimitando claramente las diferentes capas entre ellas, aplicando patrones estándar de Arquitectura para que dichas dependencias sean en muchas ocasiones basadas en abstracciones y no referenciando una capa directamente a la otra.

Cada capa de la aplicación contendrá una serie de componentes que implementan la funcionalidad de dicha capa. Estos componentes serán cohesivos internamente (dentro de la misma capa de primer nivel), pero algunas capas (como las capas de Infraestructura/Tecnología) estarán débilmente acopladas con el resto de capas para poder potenciar las pruebas unitarias, mocking, la reutilización y finalmente que impacte menos al mantenimiento.

Por ende la vista de Arquitectura Lógica es un modelo de alto nivel de la relación existente entre las capas. A partir de este modelo se presenta el modelo de implementación de arquitectura de aplicación. La arquitectura extendida puede verse de la siguiente manera:

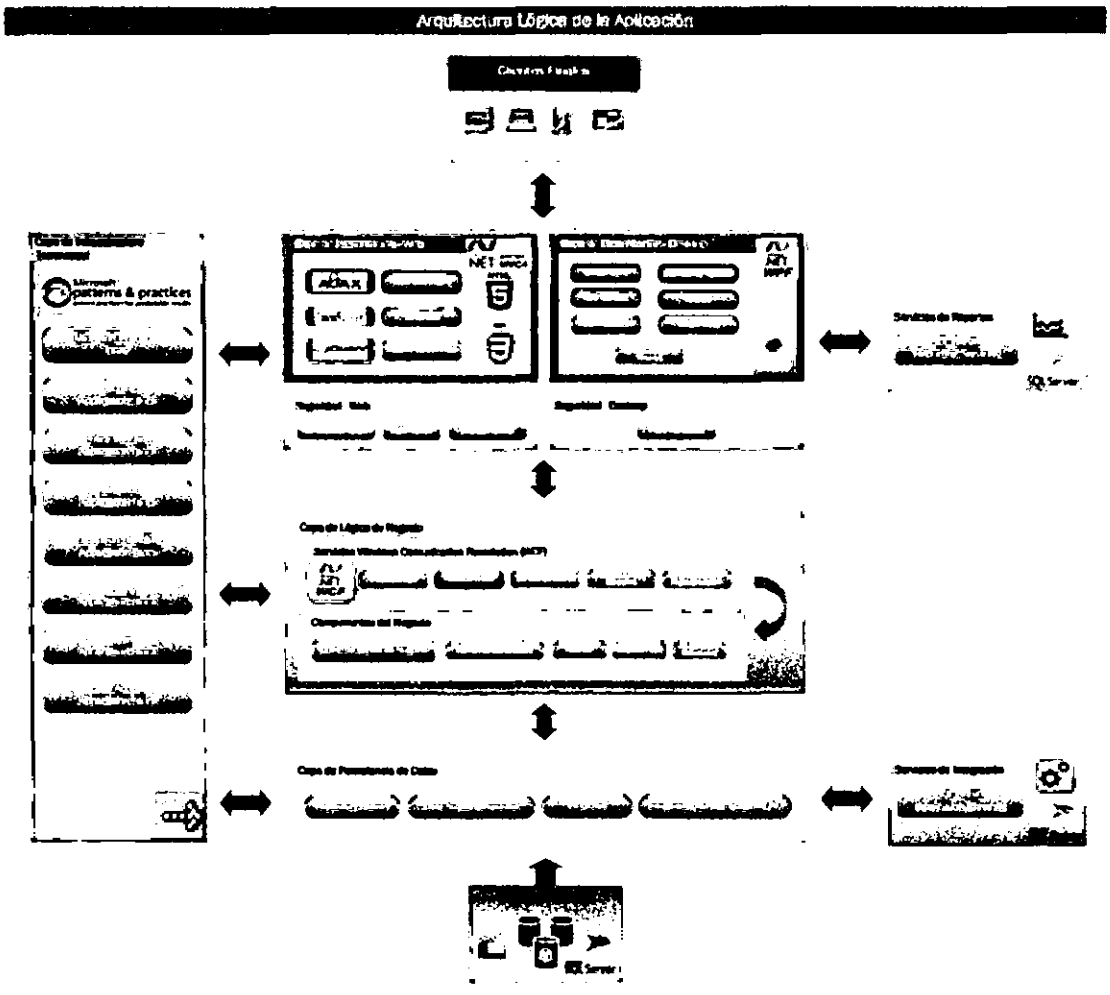


Figura 4.3.2. Arquitectura Lógica Detallada de Aplicación
Fuente: Elaboración Propia

Implementación de Arquitectura Lógica de Aplicación

La implementación de la Arquitectura Lógica de Aplicación debe estar basada en las siguientes capas:

- ✓ Capa de Presentación
- ✓ Capa de Servicios
- ✓ Capa de Negocio
- ✓ Capa de Acceso a Datos
- ✓ Capa de Datos
- ✓ Capa Transversal

El presente proyecto de investigación se centrará, como se dijo anteriormente sobre la capa de negocio, para modelar las entidades de dominio, la capa de datos, para modelar la base de datos partiendo del modelo de entidades de dominio y la capa de acceso a datos que servirá como puente entre ambas capas.

Capa de Presentación

La arquitectura propuesta por Microsoft en lo referente a la capa de presentación contiene los componentes que implementan la pantalla del usuario interfaz y gestión de la interacción del usuario, el framework base considera el diseño de la capa de presentación para aplicaciones web y también para aplicaciones de escritorio.

Esta capa incluye los controles de entrada del usuario y pantalla, además de los componentes que organizan la interacción del usuario. La figura muestra cómo la capa de presentación se inscribe en una arquitectura de aplicación común.

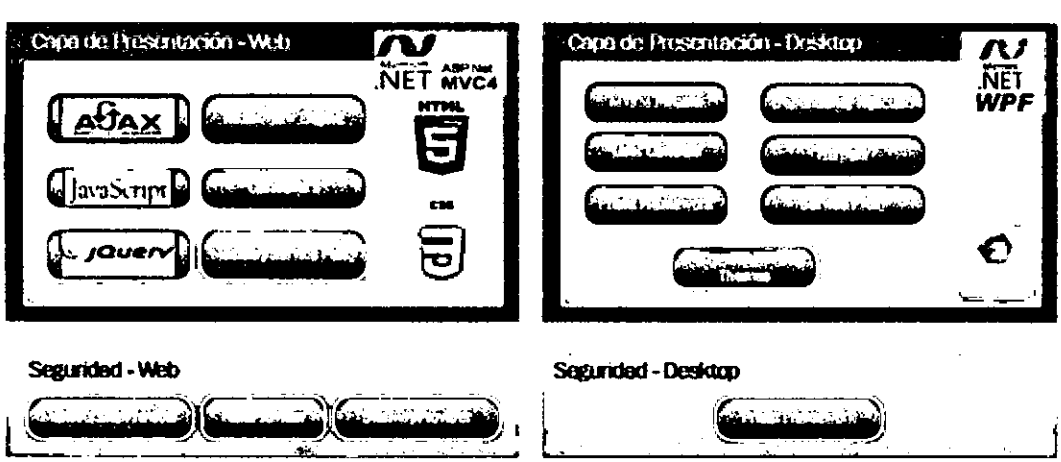


Figura 4.3.3. Capa de Presentación
Fuente: Elaboración Propia

Capa de Presentación Web:

El siguiente gráfico describe los componentes que conforman la capa de presentación para el desarrollo de aplicaciones.

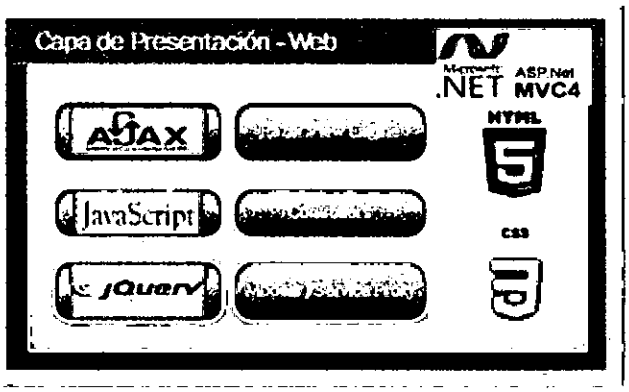


Figura 4.3.4. Capa de Presentación Web
Fuente: Elaboración Propia

Seguridad Web:

Según el siguiente gráfico, se utiliza la seguridad de ASP.NET en unión con la seguridad de Microsoft Internet Information Services (IIS) ya que incluye servicios de autenticación y autorización para implementar el modelo de seguridad en la solución de la aplicación. A continuación se detalla los mecanismos de seguridad a utilizar en el framework:

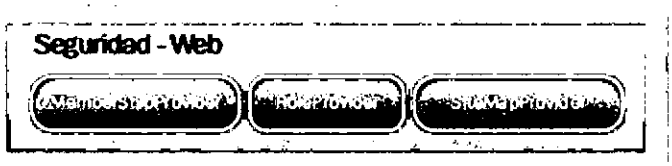


Figura 4.3.5. Seguridad de la Capa de Presentación Web
Fuente: Elaboración Propia

Capa de Presentación Desktop:

El siguiente gráfico describe los componentes que conforman la capa de presentación para el desarrollo de aplicaciones.

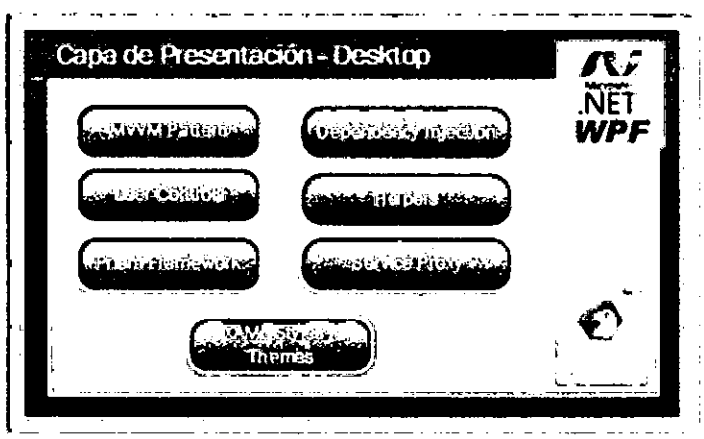


Figura 4.3.6. Capa de Presentación Desktop
Fuente: Elaboración Propia

Seguridad Desktop:

A continuación se detalla los mecanismos de seguridad a utilizar en la capa de presentación de escritorio:

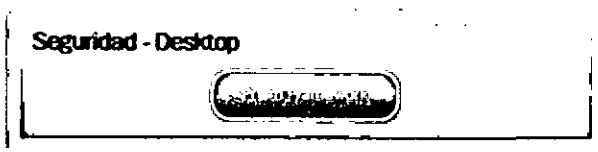


Figura 4.3.7. Seguridad de la Capa de Presentación Desktop
Fuente: Elaboración Propia

La capa de presentación se podrá adaptar para cualquier tipo de cliente:

- ✓ Cliente Web (ASP, MVC, Silverlight)
- ✓ Cliente Móvil (Windows Phone o Android)
- ✓ Cliente de Escritorio (Windows Form, WPF, Office Business Applications)

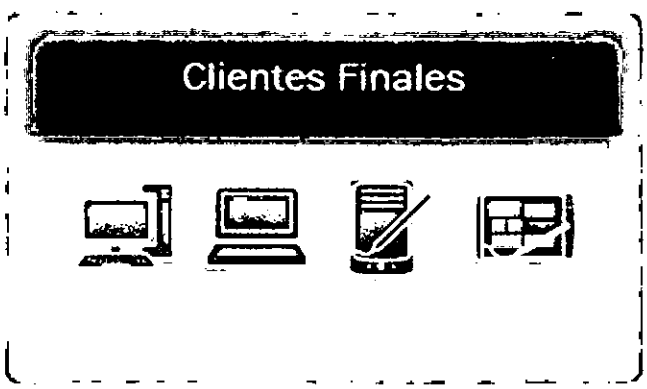


Figura 4.3.8. Clientes Finales
Fuente: Elaboración Propia

Capa de Negocio

La capa de negocio es el "puente" entre un usuario y los servicios de datos. Responden a peticiones del usuario (u otros servicios de negocios) para ejecutar una tarea de este tipo. Cumplen con esto, aplicando procedimientos formales y reglas de negocio a los datos relevantes. Cuando los datos necesarios residen en un servidor de bases de datos, garantizan los servicios de datos indispensables para cumplir con la tarea de negocios o aplicar su regla. Esto aísla al usuario de la interacción directa con la base de datos.

La definición de lógica de negocio se implementara bajo un arquetipo Orientado a Servicios. Bajo estas consideraciones es necesario implementar un patrón "Service Interface" que defina la creación inicial de contratos a manera de entidades y que estas sean las que viajen a lo largo de toda la interacción entre los clientes y el servicio.

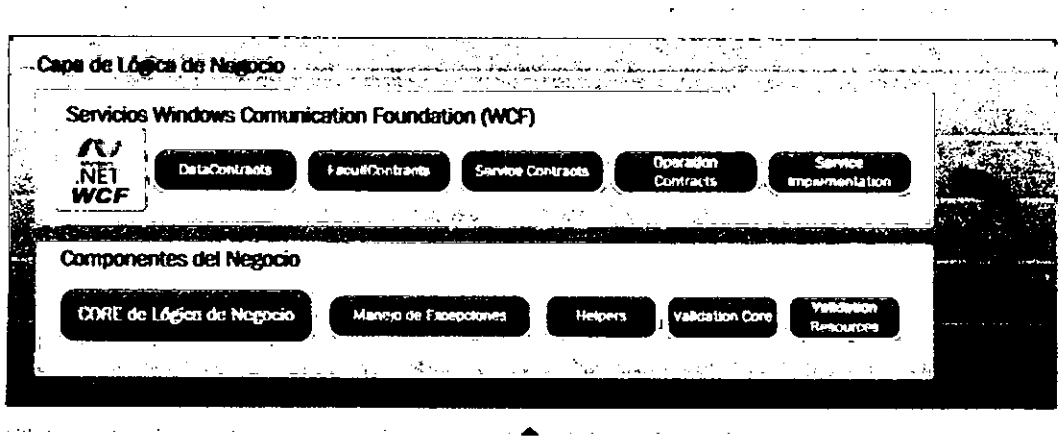


Figura 4.3.9. Capa de Negocio
Fuente: Elaboración Propia

Capa de Servicios Externos

Servicios de Reportes:

Dentro de la solución que brinda el framework de la Aplicación se utilizara "SQL Server Reporting Services" que es un componente de Microsoft SQL Server utilizado para la generación de reportes. Permitiendo que el autor del reporte defina los datos y la manera de presentación de estos. En esta etapa normalmente hay que definir conexiones a los distintos orígenes de datos para ver de dónde obtener los resultados que debe reflejar el reporte.

Figura 4.3.10. Capa de Servicios Externos - Servicios de Reportes
Fuente: Elaboración Propia

Servicios de Integración:

Dentro de la solución de APLICACIÓN se utilizara "SQL Server Integration Services (SSIS)" que es un componente de Microsoft SQL Server utilizado para migración de datos. SSIS es una plataforma para la integración de datos y sus de flujos de trabajo. Permite mover datos de origen a destino sin modificar los datos. Se pueden importar datos de fuentes diferentes a SQL Server.

Figura 4.3.11. Capa de Servicios Externos – Servicios de Integración
Fuente: Elaboración Propia

Capa de Persistencia de Datos

Esta capa posee toda la lógica de acceso e interacción con las diferentes fuentes de datos involucradas en la solución del sistema.

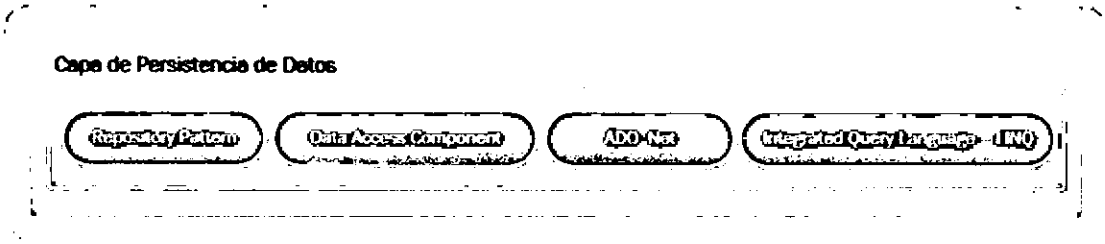


Figura 4.3.12. Capa de Infraestructura de Persistencia de Datos
Fuente: Elaboración Propia

Estos componentes proveen el acceso a los datos que está ubicado dentro de los límites del sistema y datos expuestos por otros subsistemas. El proyecto se centra en esta capa, especialmente en el componente de acceso a Datos, para el cual se utilizará el Patrón de diseño Repository, el cual se implementará usando librerías ADO .Net y librerías Enterprise Library especificadas en la sección de Infraestructura transversal:

Figura 4.3.13. Objetos de la capa de persistencia de datos a generar
Fuente: Elaboración Propia

Servicio de Acceso a Datos: Estos componentes manejan la lógica requerida para acceder a los datos almacenados. De esta manera la funcionalidad de acceso a datos se centraliza y permite fácil configuración.

Los siguientes componentes que comúnmente se encuentran en esta capa se describen a continuación:

• **Patrón Repository:** El diseño del modelo de repositorio es uno de los patrones de diseño más útiles y más ampliamente aplicables que se han inventado. Cualquier aplicación tiene que trabajar con persistencia y con algún tipo de lista de materiales. En pocas palabras, un repositorio es un mediador entre el dominio de la aplicación y los datos que le dan persistencia.

• **Componentes de Acceso a Datos:** Es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, en este caso SQL Server.

• **ADO .Net** Es un conjunto de librería que se utilizará para el acceso al repositorio de datos de SQL Server 2012.
• **Lenguaje Integrado de Consulta (LINQ):** Define operadores de consulta estándar que permiten filtrar, enumerar y crear proyecciones de varios tipos de colecciones usando la misma sintaxis. Tales colecciones pueden incluir vectores (arrays), clases enumerables, XML, conjuntos de datos desde bases de datos relacionales y orígenes de datos de terceros.

Adicionalmente se utilizará el gestor de base de datos SQL Server 2012

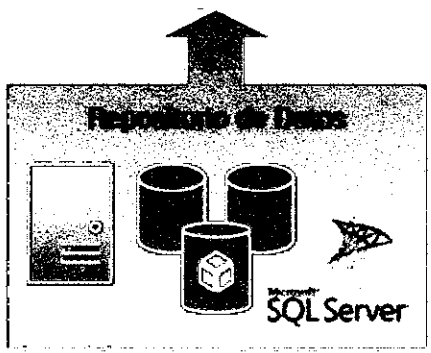


Figura 4.3.14. Capa de Datos – SQL Server 2012
Fuente: Elaboración Propia

Capa Transversal

En esta capa se pueden apreciar componentes globales que se caracterizan porque pueden ser usados a través de todas las capas de la arquitectura.

Enterprise Library 6.0: Es una colección de componentes de software reutilizables (bloques de aplicación) diseñados para ayudar a los desarrolladores de software empresariales con el desarrollo de cuestiones transversales comunes como el registro, validación, acceso a datos, manejo de excepciones, y muchos otros.

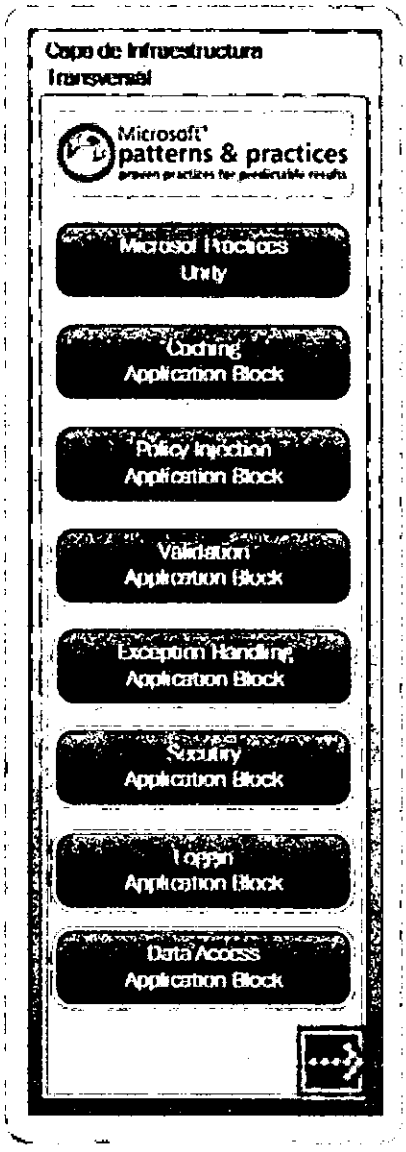


Figura 4.3.15. Capa de Infraestructura Transversal
Fuente: Elaboración Propia

Tecnologías a Utilizar

Las tecnologías que se van a considerar dentro de este marco de trabajo son las siguientes:

Plataforma y entorno de desarrollo
- ✓ .NET Framework 4.0
- ✓ Visual Studio 2012

Capa de Datos
- ✓ SQL Server 2012

Capa Transversal (Capa de Acceso a Datos)
- ✓ Enterprise Library 6.0

CAPÍTULO V: IDENTIFICACIÓN DE REQUERIMIENTOS DE MEJORAS

5.1. Identificación de mejoras en la arquitectura del proyecto

Se han identificado las siguientes mejoras sobre la arquitectura del proyecto:

Diseño y generación de la arquitectura del proyecto

La creación y diseño de la solución y los proyectos de librerías que necesita la arquitectura de N-Capas orientado al dominio se hacen manualmente cada vez que se inicia un nuevo proyecto. La estructura de carpetas y archivos necesarios para implementar la solución de negocio se copia desde soluciones de proyectos anteriores.

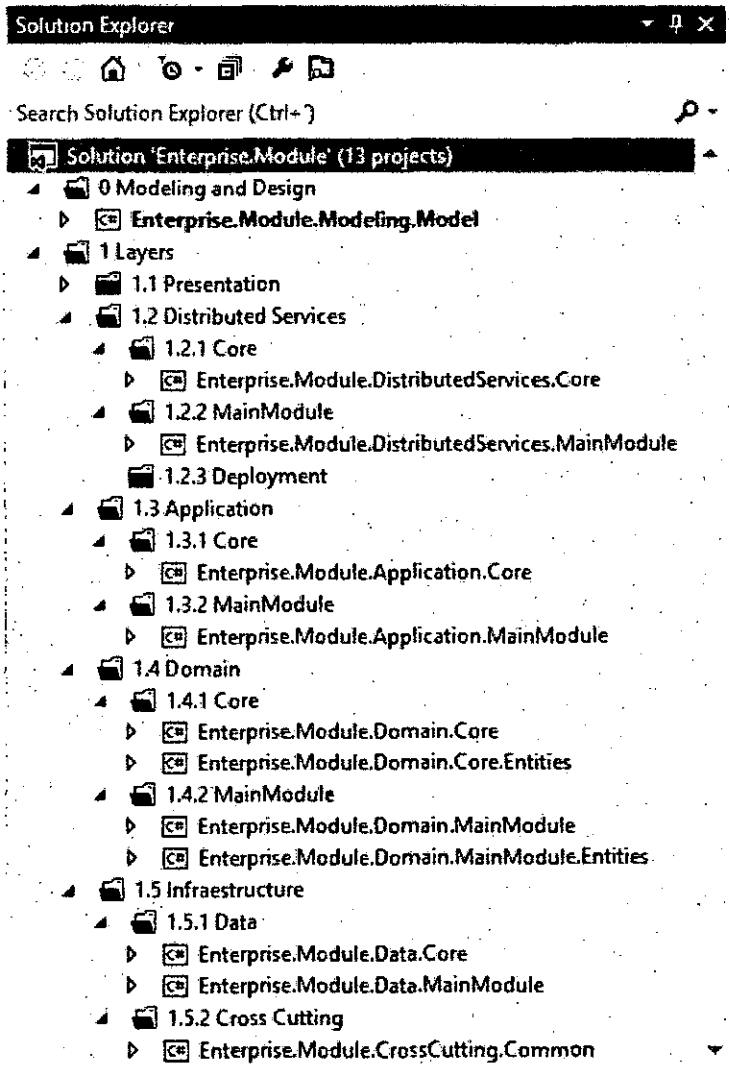


Figura 5.1.1. Estructura de la solución N-Capas
Fuente: Elaboración Propia

La mejora a implementar se basa en crear en una plantilla de Visual Studio (.vstemplate) para generar la estructura mostrada en el gráfico anterior, con los archivos necesarios para que se empiece a programar la aplicación.

5.2. Identificación de mejoras en la capa de datos

Se han identificado las siguientes mejoras sobre la capa de datos o base de datos:

Diseño y generación de Tablas:

Si bien las tablas se pueden diseñar desde el editor de tablas de SQL Management 2012 siguiendo los estándares de base de datos propios de la empresa consultora y basándose en las clases de dominio, se identificó que sería mejor basar el diseño de la base de datos partiendo de un modelo de entidades de dominio definidas. Estas entidades de dominio estarían plasmadas en un modelo diseñado en un diagrama DSL propuesto en el presente proyecto que incluye metadata sobre columnas relacionadas a cada una de las propiedades de la entidad.

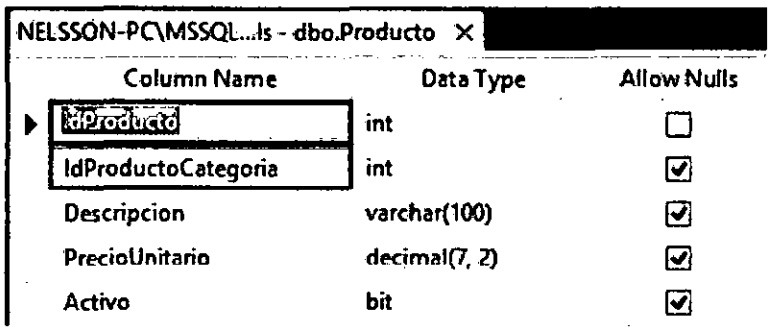


Figura 5.2.1. Creación manual de tablas mediante SQL Management
Fuente: Elaboración Propia

De la misma forma las relaciones entre tablas se pueden mapear con el modelo de entidades mediante relaciones de agregación:

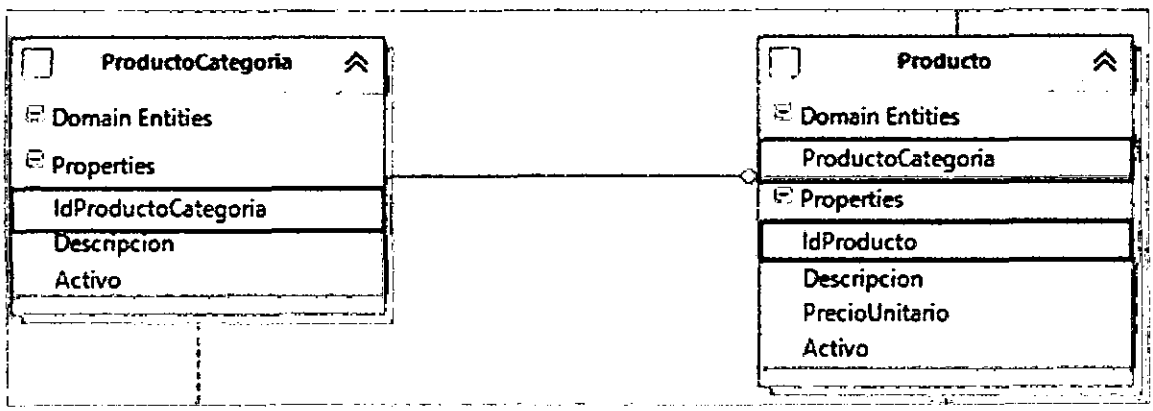


Figura 5.2.2. Relación entre entidades para generación de relaciones entre tablas
Fuente: Elaboración Propia

El modelo de transformación para la generación de scripts de creación de tablas sería el siguiente:

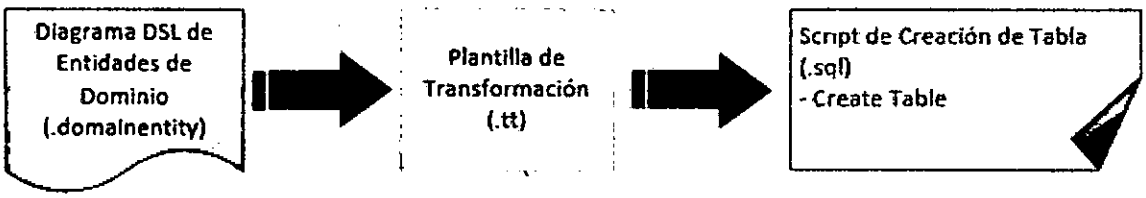


Figura 5.2.3. Generación de scripts de creación de tablas
Fuente: Elaboración Propia

Diseño y generación de Procedimientos Almacenados:

Cada vez que se crea una tabla se tienen que elaborar los procedimientos almacenados transaccionales sobre esta (inserción, actualización, eliminación y consulta), y dado que la metadata sobre columnas se establecería en el modelo mencionado anteriormente, estos procedimientos almacenados se podrían generar a partir de este.

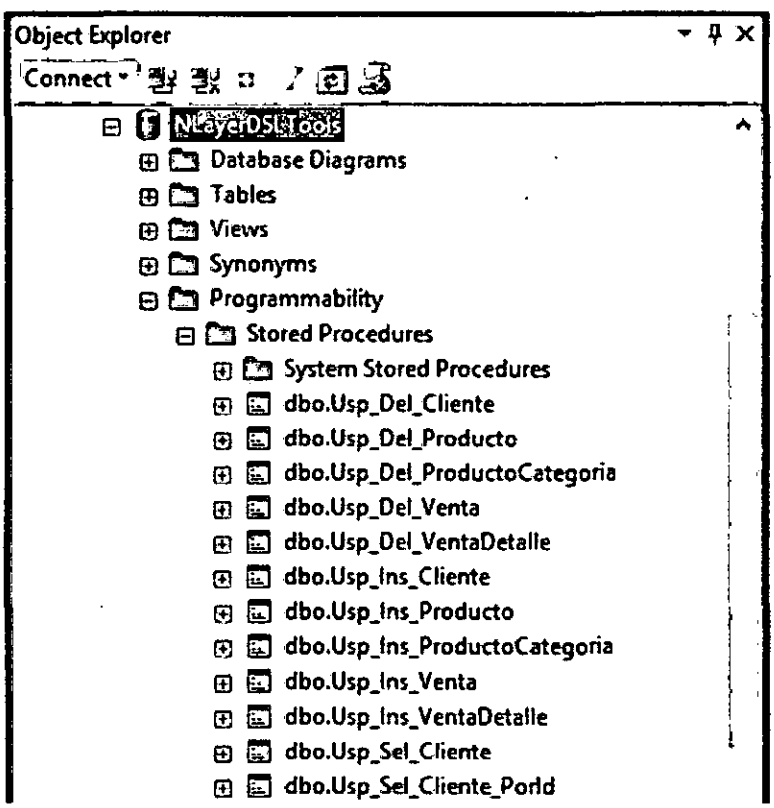


Figura 5.2.4. Procedimientos creados manualmente
Fuente: Elaboración Propia

El esquema para generar los procedimientos almacenados sería el siguiente:

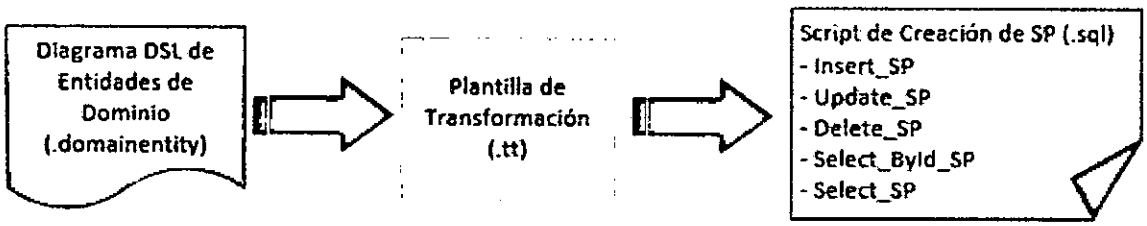


Figura 5.2.5. Generación de scripts de creación de procedimientos almacenados
Fuente: Elaboración Propia

En general las características a mejorar en el desarrollo del proyecto en cuanto a base de datos serían las siguientes:

- ✓ Generación de scripts de creación de tablas en base al modelo de entidades del dominio.
- ✓ Generación de scripts de creación de procedimientos almacenados en base al modelo de entidades del dominio.

Y el modelo general de generación de objetos de base de datos por cada entidad del modelo de dominio sería el siguiente:

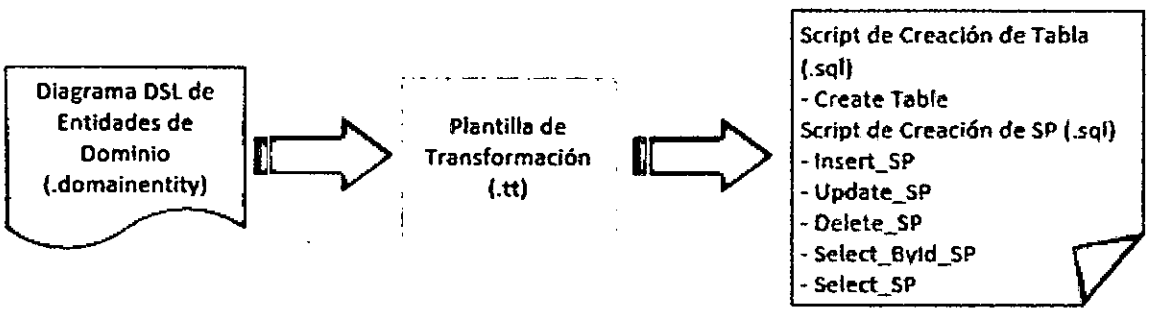


Figura 5.2.6. Generación de scripts de objetos de base de datos
Fuente: Elaboración Propia

5.3. Identificación de mejoras en la capa de entidad

Las mejoras identificadas para la capa de entidad son las siguientes:

Diseño y generación de Entidades de Dominio

La creación de entidades de dominio durante el desarrollo de proyectos de software mayormente se hace de forma manual, es decir, que se agregan clases y se programan de acuerdo a las necesidades del producto. Estas entidades también se pueden diseñar en un diagrama DSL especificando las características de la entidad como propiedades de la entidad. También se puede diseñar las relaciones entre clases mediante relaciones de agregación, esto permite tener una idea más general de las entidades del modelo.

Producto.cs

Enterprise.Module.Domain.Core.Entities.Producto — ProductoCategoria

```
1  namespace Enterprise.Module.Domain.Core.Entities
2  {
3      public partial class Producto
4      {
5
6          public ProductoCategoria ProductoCategoria { get; set; }
7
8          public System.Int32 IdProducto { get; set; }
9
10         public System.String Descripcion { get; set; }
11
12         public System.Double PrecioUnitario { get; set; }
13
14         public System.Boolean Activo { get; set; }
15
16     }
17 }
18
```

Figura 5.3.1. Estructura de una clase de entidad de dominio
Fuente: Elaboración Propia

El modelo de entidades del dominio se vería de esta forma:

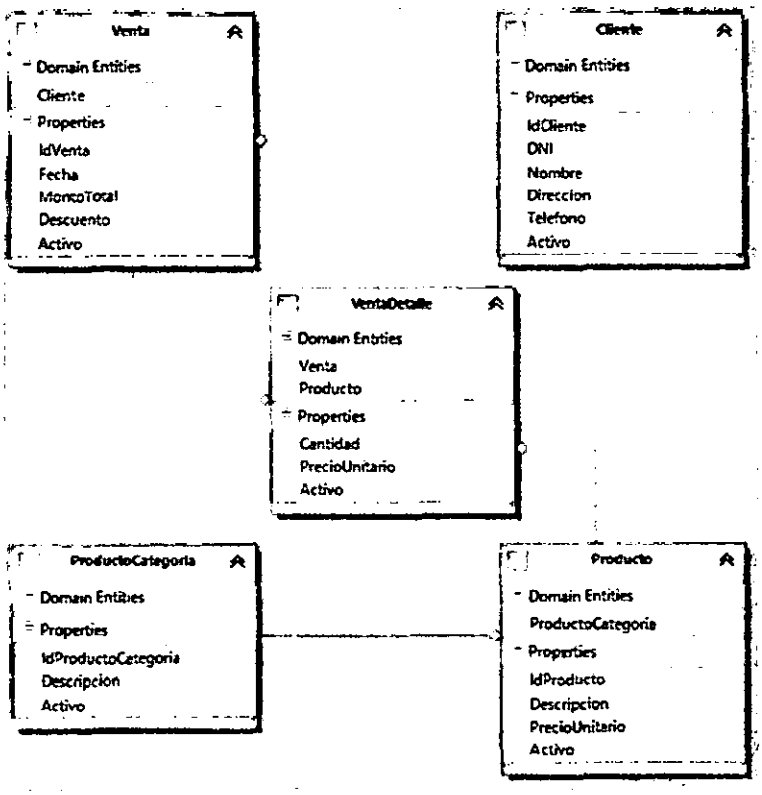


Figura 5.3.2. Diseño entidades mediante un diagrama DSL
Fuente: Elaboración Propia

Debido a que la metadata sobre propiedades y relaciones entre entidades se encuentran en el modelo DSL, las entidades se pueden generar mediante una plantilla de transformación. La generación de entidades por cada entidad del modelo se haría de la siguiente forma:

Figura 5.3.3. Generación de clases de entidades
Fuente: Elaboración Propia

Diseño y generación de Listas y/o Colecciones de Entidades de Dominio

De la misma forma en la que se crean manualmente las clases de entidades, se crean también las colecciones o listas de entidades para la recuperación de varias entidades desde la base de datos. La mejora a implementar consistiría en implementar el diseño de colecciones de entidades en el diagrama DSL.

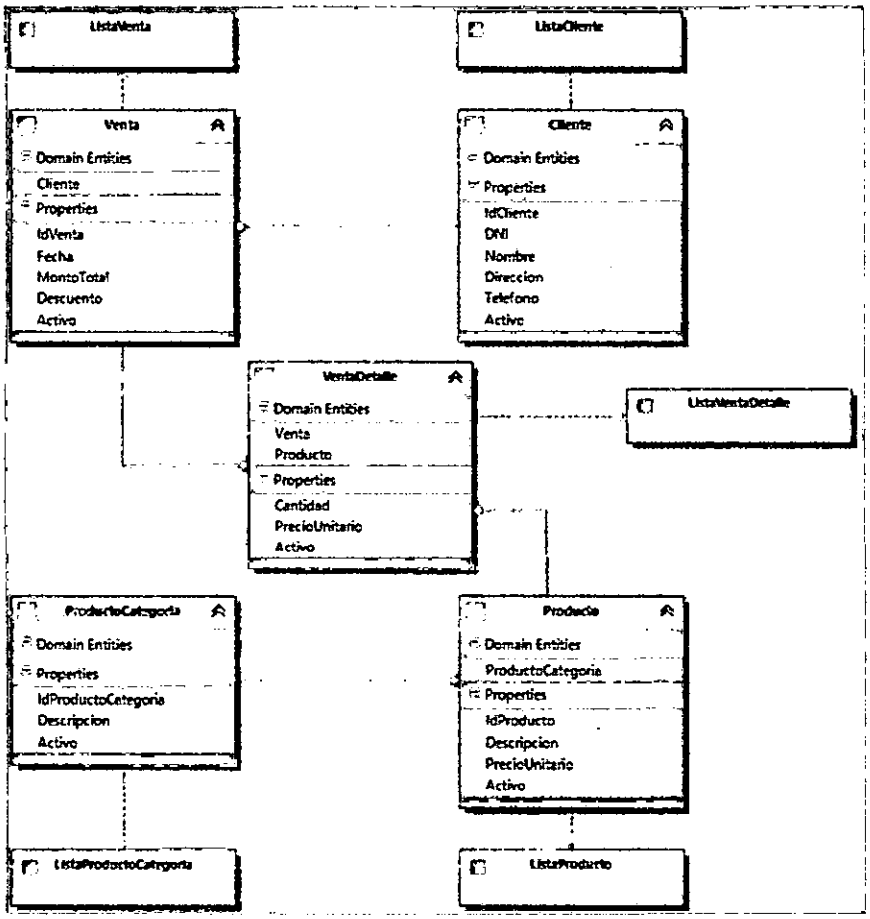


Figura 5.3.4. Diseño de colecciones de entidades en el diagrama DSL
Fuente: Elaboración Propia

La estructura de la clase de colección de entidades es la siguiente:

ListaProducto.cs

Enterprise.Module.Domain.Core.Entities.ListaProducto

```
1  namespace Enterprise.Module.Domain.Core.Entities
2  {
3      public partial class ListaProducto : System.Collections.Generic.List<Producto>
4      {
5
6      }
7  }
8
```

Figura 5.3.5. Estructura de una clase de colección de entidades de dominio
Fuente: Elaboración Propia

Y la generación de clases de colecciones de entidades utilizando una plantilla es la siguiente:

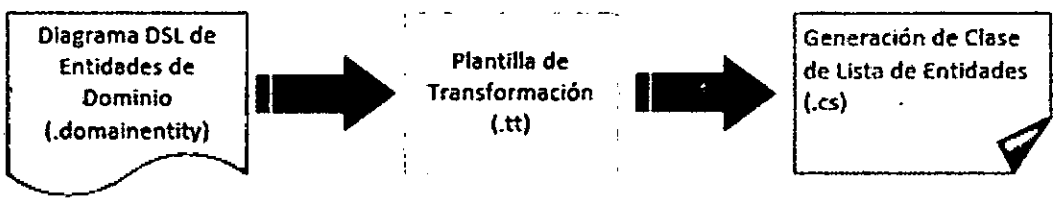


Figura 5.3.6. Generación de clases de colecciones de entidades
Fuente: Elaboración Propia

En resumen, las mejoras sobre la capa de entidad son las siguientes:

- ✓ Diseño gráfico y generación de clases de entidades de dominio.
- ✓ Diseño gráfico y generación de clases de colecciones de entidades de dominio.

5.4. Identificación de mejoras en la capa de acceso a datos

Las mejoras identificadas en la capa de acceso a datos son las siguientes:

Generación del componente de acceso a datos

Una de los componentes que más tiempo de desarrollo consume, es la programación de los componentes de acceso a datos. Debido a que el modelo de datos y el modelo de entidad de dominio se encuentran separados es muy común encontrar errores de programación en los tipos de datos, parámetros de procedimientos almacenados y mapeo de campos sobre las consultas.

Ya que se utiliza el Patrón Repository en el marco de N-Capas, se trata a una tabla simplemente como un repositorio de datos a los que se le implementan los métodos de acceso correspondientes de inserción, actualización, eliminación y consulta; todos los componentes de acceso a datos tienen la misma estructura y por lo tanto pueden generarse mediante una plantilla que lea el modelo de entidades de dominio en el que se encuentran la metadata.

La generación de código de acceso a datos mediante una plantilla sería de la siguiente forma:

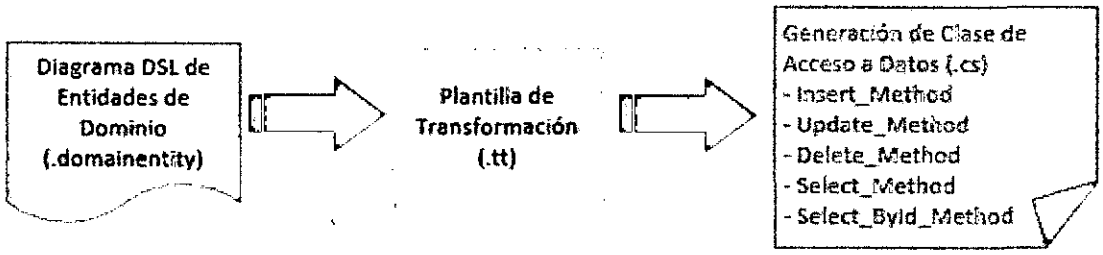


Figura 5.3.7. Generación de clases de acceso a datos
Fuente: Elaboración Propia

La mejora consiste en usar el modelo de entidades de dominio para generar el componente de acceso a datos.

CAPÍTULO VI: DISEÑO DE LA SOLUCIÓN PROPUESTA

6.1. Diseño de la arquitectura del proyecto

La arquitectura del proyecto se basará en la arquitectura de N-Capas orientada al dominio propuesta por Microsoft y plasmada en la sección 4.3 del presente documento.

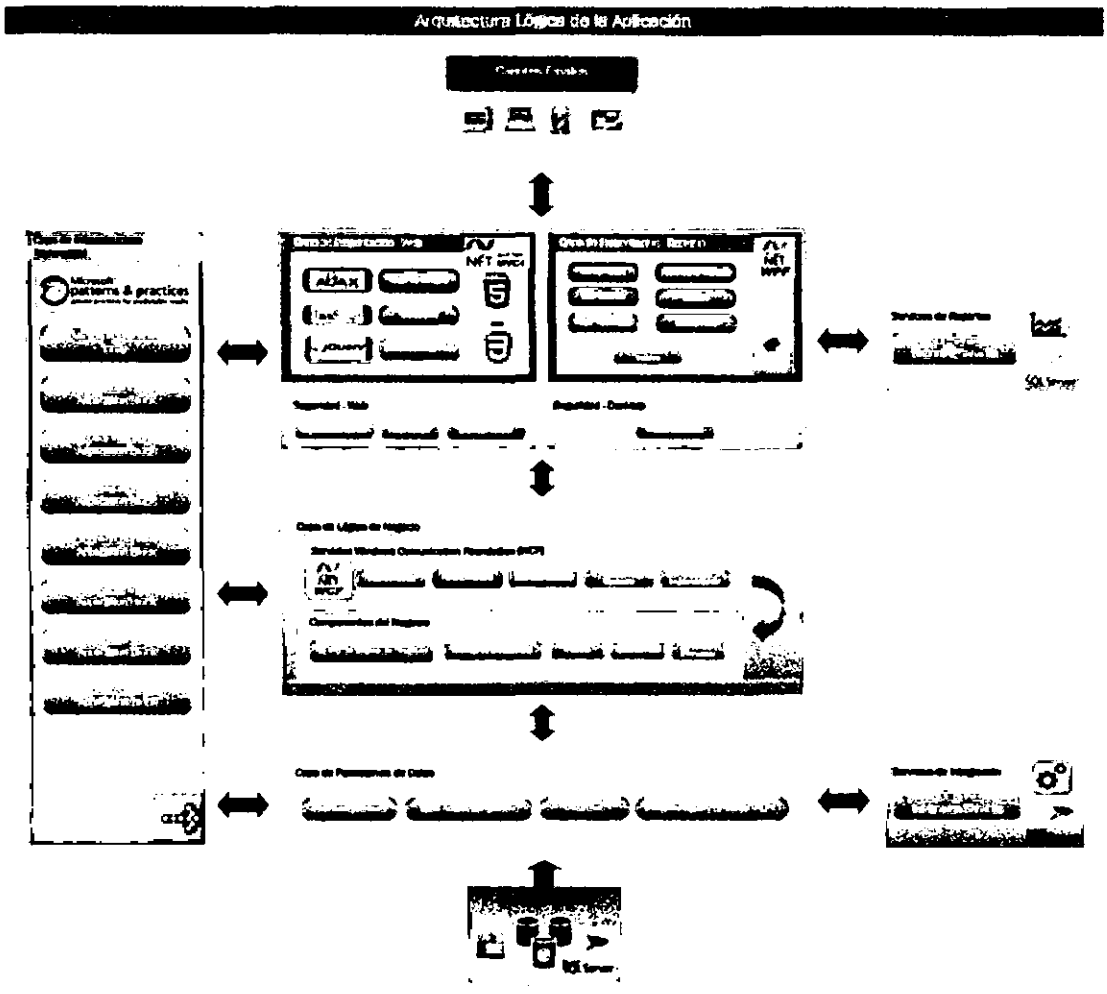


Figura 6.1.1. Arquitectura Lógica detalla de Aplicación
Fuente: Elaboración Propia

6.2. Diseño de la plantilla de la solución

Para implementar la solución de Visual Studio 2012 de la arquitectura lógica del proyecto se va a tener en cuenta la siguiente estructura:

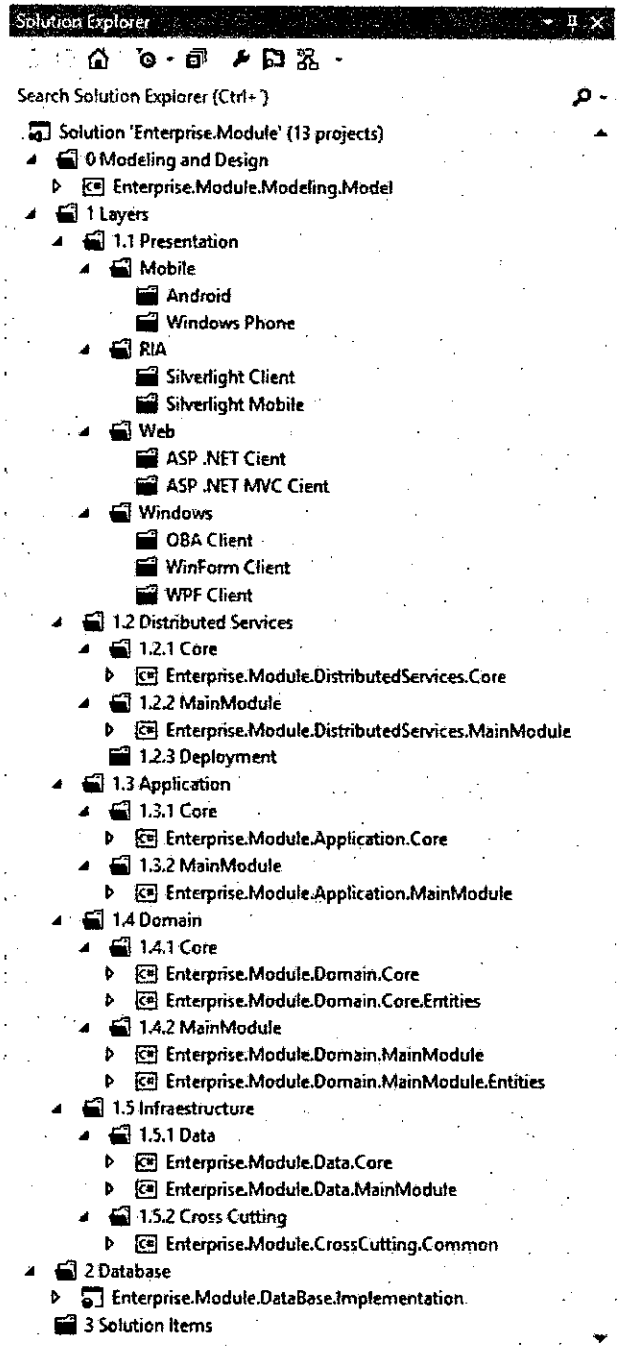


Figura 6.2.1. Estructura de la solución de la arquitectura N-Capas
Fuente: Elaboración Propia

El nombre de la solución dependerá del nombre que establezca el usuario al momento de crearla. La nomenclatura del nombre de la solución es la siguiente: **[Enterprise.Module]**

Ya que normalmente se crea la solución basándose en el nombre de la empresa para la que se crea (Enterprise), seguida por el módulo que se va a implementar (Module) separados por un punto ".". Este nombre es tomado en cuenta para generar los nombres de los proyectos de infraestructura de la arquitectura.
La estructura de cada carpeta y proyecto de la solución se describe a continuación:

✓ 0 Modeling and Design

Esta carpeta de solución contiene el proyecto de modelado y diseño de entidades de dominio, es en este proyecto donde se encuentra el diagrama DSL sobre el cual se van a modelar las entidades y listas de entidades y las relaciones correspondientes. La estructura del folder es la siguiente:

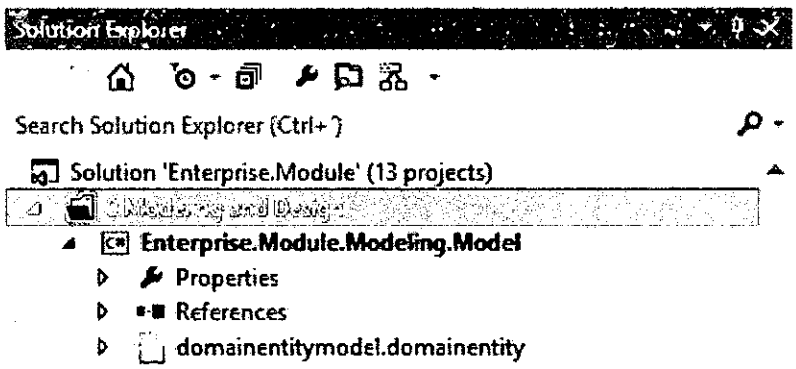


Figura 6.2.2. Estructura de la carpeta de modelado y diseño
Fuente: Elaboración Propia

***[Enterprise.Module].*Modeling.Model:** Este proyecto es de tipo librería de clases (Class Library) y contiene por defecto un archivo denominado *domainentitymodel.domainentity* que es el archivo de modelado DSL.

✓ 1 Layers

Esta carpeta de solución contiene los proyectos de las diferentes capas de la arquitectura marco de N-Capas, aquí se encuentran tanto los proyectos del Front-End, para la implementación de aplicaciones de los diferentes tipos de clientes finales como aplicaciones de escritorio, RIA, Web y Móvil también contiene los proyectos del Back-End de la aplicación tales como los proyectos de entidades de dominio, acceso a datos, servicios distribuidos y proyectos comunes usados para la capa de infraestructura transversal. La estructura de la carpeta es la siguiente:

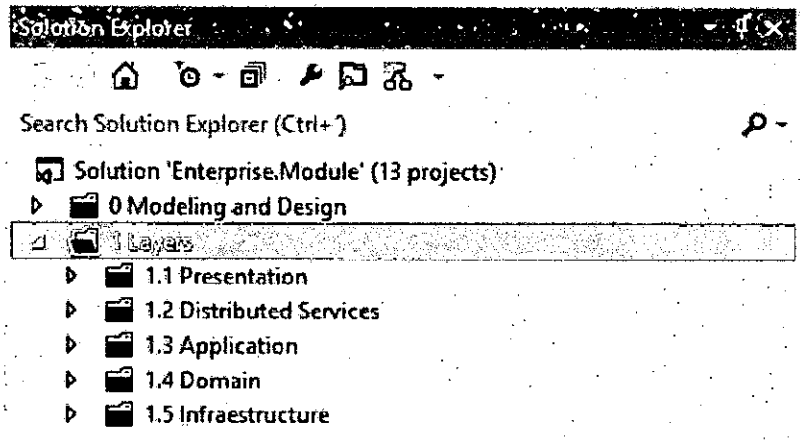


Figura 6.2.3. Estructura de la carpeta de modelado y diseño
Fuente: Elaboración Propia

1.1 Presentation

Esta carpeta tiene una estructura de carpetas para el desarrollo de la capa de presentación del proyecto, aquí se van a albergar los proyectos para los clientes finales. La estructura es la siguiente:

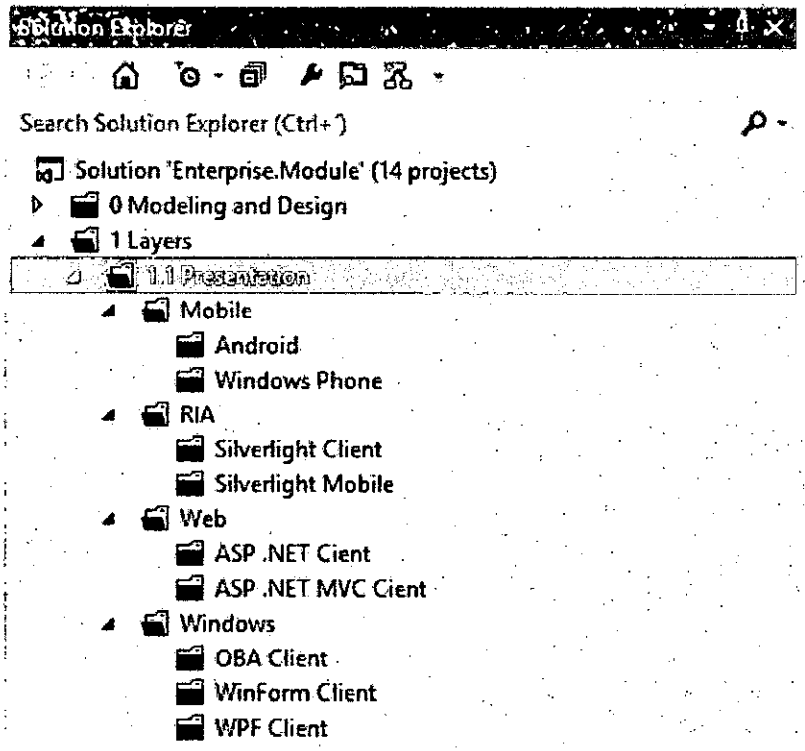


Figura 6.2.4. Estructura de la carpeta de presentación
Fuente: Elaboración Propia

La carpeta contiene secciones para poder agregar proyectos de cuatro tipos:

Mobile: Aquí se pueden agregar proyectos de tipo **Android** (Utilizando Xamarin, que es un complemento de Visual Studio para esta plataforma) o proyectos de **Windows Phone** para dispositivos móviles windows.

RIA: Aquí se agregarán los proyectos de tipo Rich Internet Applications (Aplicaciones de Internet Enriquecidas), para el caso de la plataforma Microsoft se utilizan aplicaciones de tipo **Silverlight Client**, que son aplicaciones Silverlight tradicionales para web, o aplicaciones **Silverlight Mobile** adaptadas para terminales móviles.

Web: Aquí se pueden agregar proyectos de tipo **ASP .Net Client** para proyectos Web con motor de vistas ASP o proyectos **ASP .Net MVC Client** para clientes con el patrón MVC y motor de vistas Razor.

Windows: Esta carpeta está destinada papa proyectos de tipo escritorio, ya sea proyectos **WinForm Client, WPF Client** para aplicaciones de escritorio o extensiones para Office, **OBA Client** (Office Business Aplications).

1.2 Distributed Services

Esta carpeta está destinada a contener los proyectos de tipo librería (Class Library) que implementan Servicios Web (WCF Services) y no deben tener implementada ninguna regla de negocio sino simplemente hacer llamados a la capa de aplicación, la estructura de la carpeta es la siguiente:

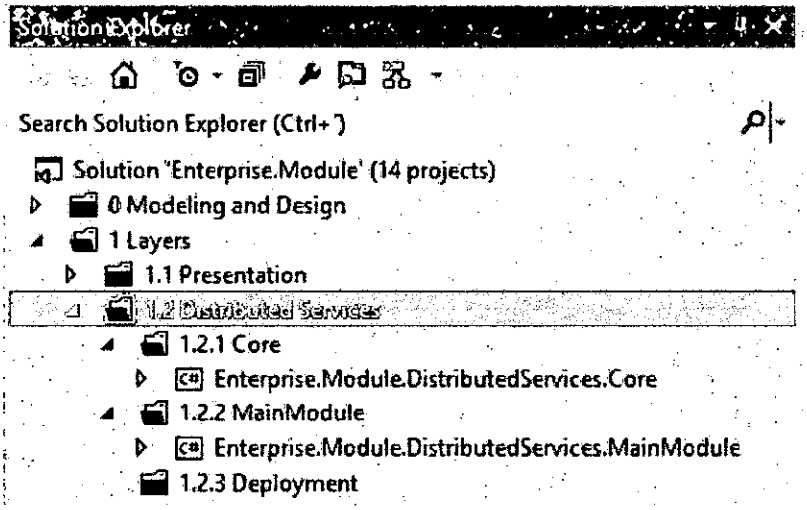


Figura 6.2.5. Estructura de la carpeta de servicios distribuidos
Fuente: Elaboración Propia

1.2.1 Core

***[Enterprise.Module].*DistributedServices.Core:** Este proyecto es de tipo librería de clases (Class Library) y va a contener el núcleo de implementación de los Servicios Web, clases necesarias para ejecutar los servicios.

1.2.2 MainModule

***[Enterprise.Module].*DistributedServices.MainModule:** Este proyecto también es de tipo librería de clases (Class Library) y contiene la implementación de cada uno de los métodos expuestos a través de Web Services del módulo principal de la aplicación.

1.2.3 Deployment

En esta carpeta está destinada a contener el proyecto de despliegue de los servicios, normalmente un proyecto web que va a ser desplegado en un servidor de aplicaciones Internet Information Services (IIS) de Windows Server.

1.3 Application

En esta carpeta va a contener los proyectos de servicios de aplicación (no son servicios web), estos servicios están expuestos hacia la capa de servicios distribuidos a manera de métodos de interfaces implementados por las clases correspondientes, estos métodos y su implementación no deben contener lógica de negocio y es la encargada de gestionar recursos de la capa de acceso a datos, envió de correo, transferencia de archivos, etc. La estructura es la siguiente:

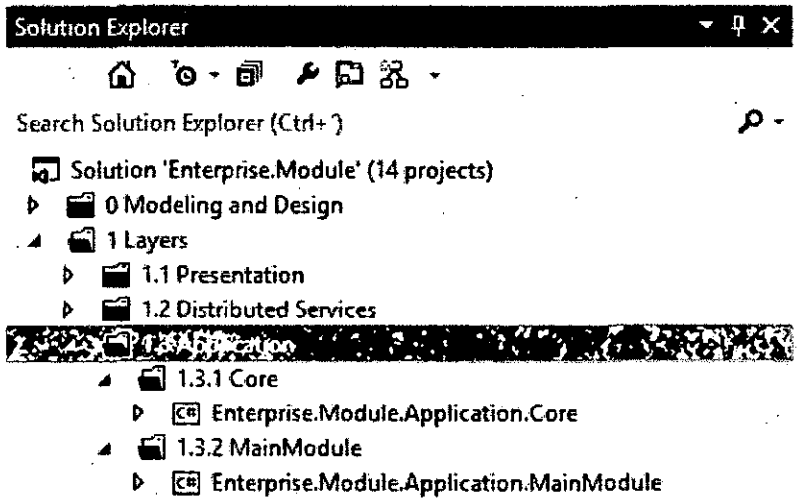


Figura 6.2.6. Estructura de la carpeta de aplicación
Fuente: Elaboración Propia

1.3.1 Core

***[Enterprise.Module].*Application.Core:** Este proyecto es de tipo librería de clases (Class Library) y contiene los servicios de aplicación, es decir las interfaces para exposición de métodos a la capa de servicios distribuidos.

1.3.2 MainModule

***[Enterprise.Module].*Application.MainModule:** Este proyecto es también de tipo librería de clases (Class Library) y contiene las clases de implementación las interfaces de los servicios de aplicación.

1.4 Domain

Esta carpeta contiene el núcleo de la aplicación, es aquí donde se encuentran las clases de entidades de dominio y las clases de lógica de negocio referente a la aplicación, de manera que esta es la capa más importante del proyecto en la que se centra la arquitectura marco orientada al dominio y la que implementa el dominio de la aplicación, la estructura es la siguiente:

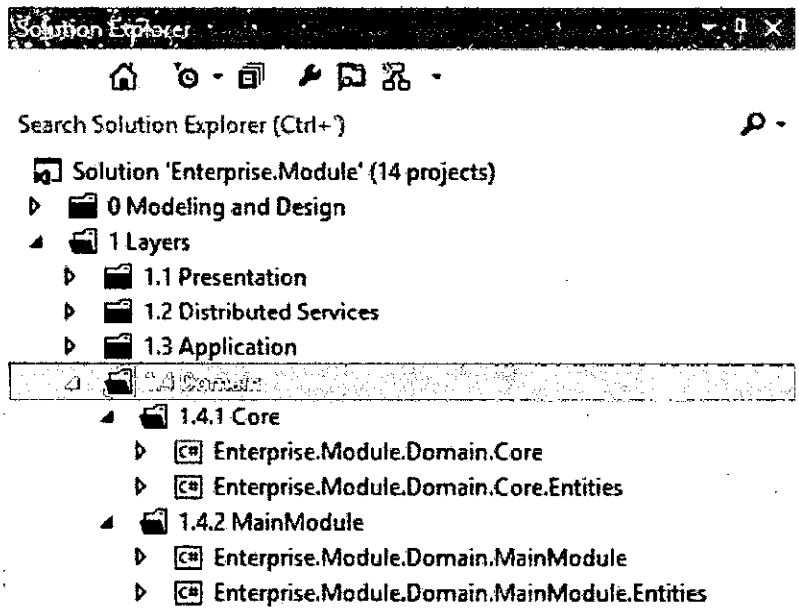


Figura 6.2.7. Estructura de la carpeta de dominio
Fuente: Elaboración Propia

1.4.1 Core

***[Enterprise.Module].*Domain.Core:** Este proyecto de tipo librería de clases contiene las clases base y las clases reutilizables en el módulo de entidades de dominio.

***[Enterprise.Module].*Domain.Core:** Este proyecto es de tipo librería de clases y contiene las clases de entidad de dominio, es en este proyecto en el que se van a generar las entidades desde el modelo de dominio especificados en el diagrama DSL.

1.4.2 MainModule

***[Enterprise.Module].*Domain.MainModule:** Este proyecto de tipo Class Library tiene clases base y clase de métodos comunes y utilitarios necesarios para implementar el módulo principal de la aplicación.

***[Enterprise.Module].*Domain.MainModule:** Este proyecto de tipo librería de clases contiene las funciones de la lógica del dominio que son expuestas a través de interfaces hacia la capa de aplicación y que contiene también las clases de implementación de las interfaces expuestas, se pueden agregar módulos de aplicación de acuerdo a las necesidades.

1.5 Infraestructure

Esta carpeta contiene la infraestructura necesaria para la persistencia y acceso a datos y también para la infraestructura transversal de la aplicación como conceptos de login, seguridad, etc. La estructura es la siguiente:

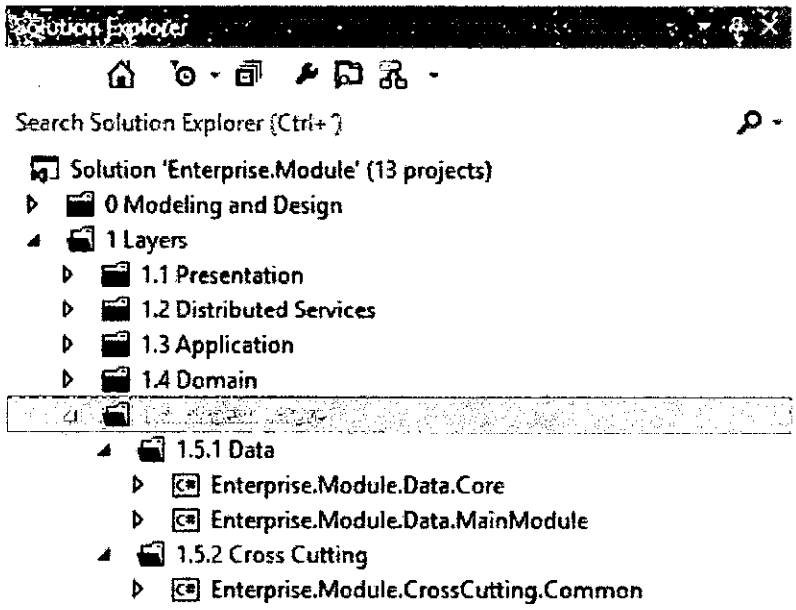


Figura 6.2.8. Estructura de la carpeta de infraestructura
Fuente: Elaboración Propia

1.5.1 Data

***[Enterprise.Module].*Data.Core:** Este proyecto de tipo librería de clases contiene las clases base y las clases reutilizables en el módulo principal de persistencia de datos.

***[Enterprise.Module].*Data.MainModule:** Este proyecto de tipo librería de clases contiene las clases de acceso a datos del módulo principal, es en este proyecto en el que se van a generar las clases de acceso a datos desde el modelo de dominio especificados en el diagrama DSL de proyecto de modelado.

1.5.2 Cross Cutting

***[Enterprise.Module].*CrossCutting.Common:** Este proyecto de tipo Class Library tiene clases comunes y utilitarias que van a ser usadas por otras capas de la arquitectura, aquí se puede establecer conceptos de seguridad, loggin, auditoría, utilitarios de envió de correo, transferencia de archivos mediante protocolo ftp y más clases que puedan ser utilizadas.

✓ 2 Database

Esta carpeta contiene un proyecto de implementación de base de datos donde se van a generar los scripts de objetos de base de datos, la estructura es la siguiente:

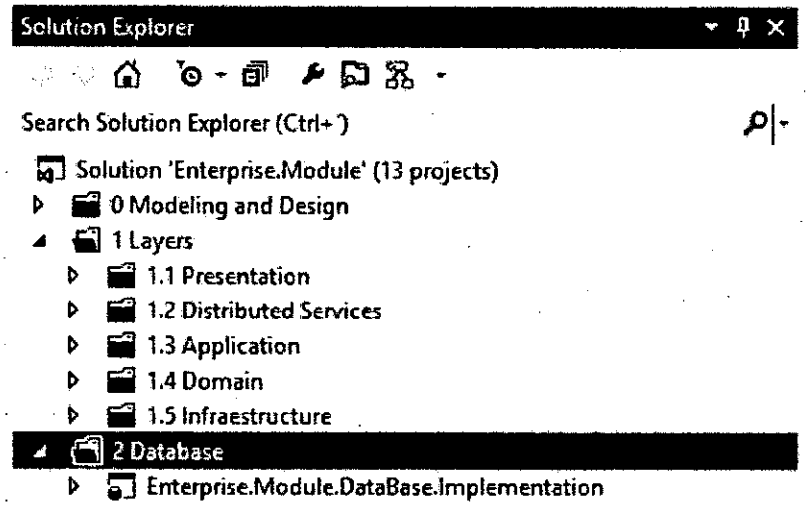


Figura 6.2.9. Estructura de la carpeta de base de datos
Fuente: Elaboración Propia

***[Enterprise.Module].*DataBase.Implementation:** Este proyecto es de tipo proyecto de base de datos y contiene inicialmente un script llamado *Objects_CreateDataBase.sql* que tiene una sentencia para crear una base de datos, este proyecto servirá como repositorio de los scripts generados para creación de tablas y procedimientos almacenados a partir del modelo especificado en el diagrama DSL del proyecto de modelado y diseño de la solución.

✓ 2 Solution Items

Carpeta está destinada a contener proyectos externos a la arquitectura y que puedan ser necesarios para implementar la aplicación a desarrollar, proyectos de archivos de recursos, archivos de contenido, etc. La estructura es la siguiente:

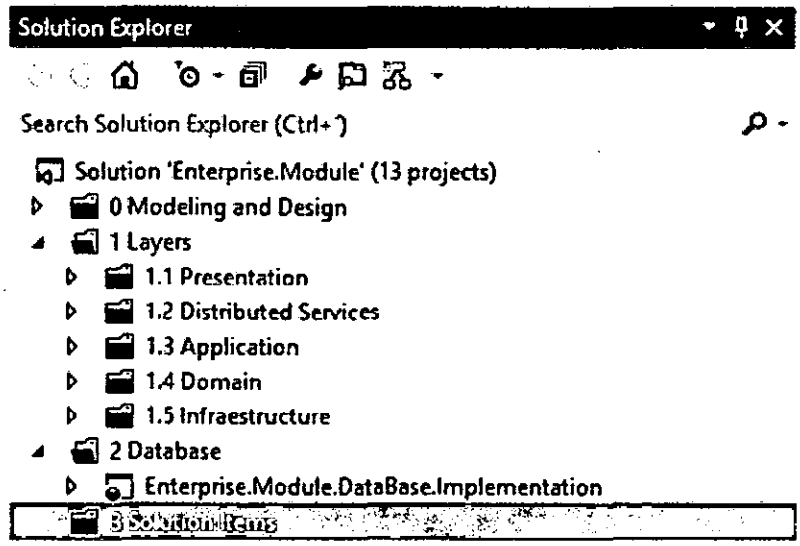


Figura 6.2.10. Estructura de la carpeta de ítems de la solución
Fuente: Elaboración Propia

6.3. Diseño del componente de entidad

El componente de entidad se establecerá en un modelo de lenguaje específico de dominio, para ello se ha diseñado el diagrama del modelo en un proyecto de extensibilidad de Visual Studio para DSL denominado *"N-Layer DSL Tools"*, la especificación del modelo se encuentra en el archivo *DslDefinition.dsl* del proyecto.

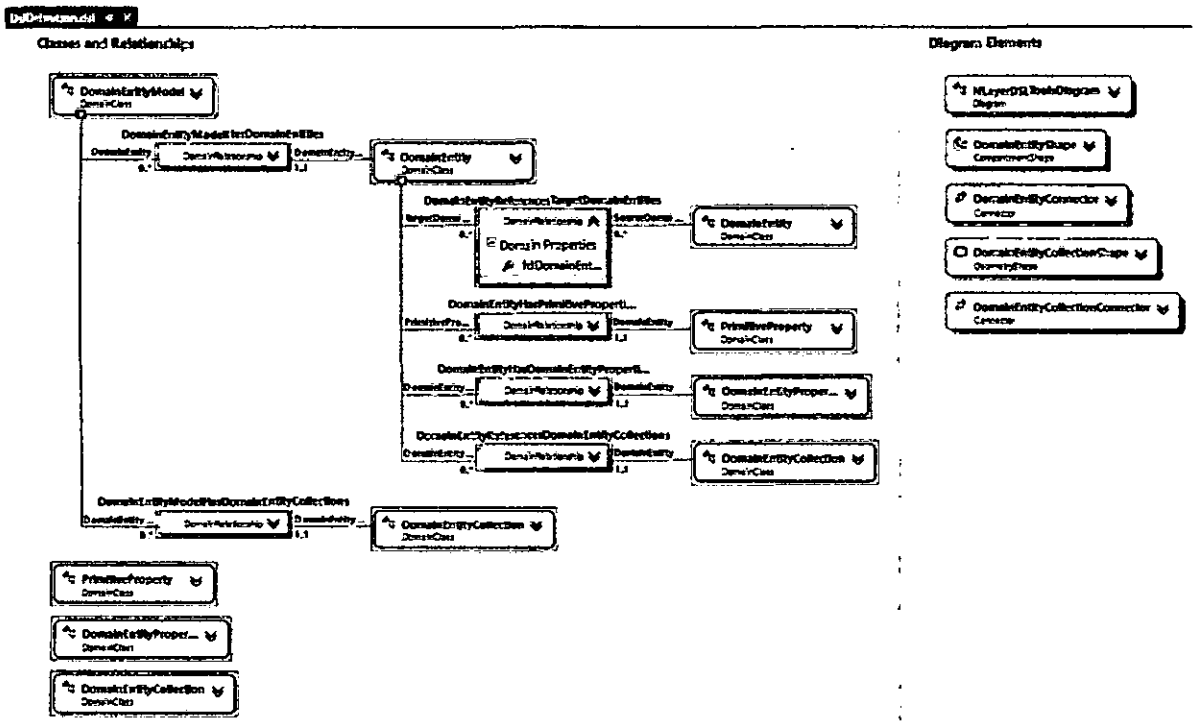


Figura 6.3.1. Diagrama de definición del dominio de entidades
Fuente: Elaboración Propia

A continuación se especifican y detallan los componentes del modelo de dominio de entidades.

Clase DomainEntityModel

Esta es la **clase base del modelo** de entidades de dominio, esta clase tiene información de los proyectos de generación de código, la estructura de la clase es la siguiente.

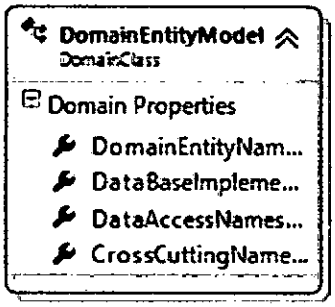


Figura 6.3.2. Clase DomainEntityModel
Fuente: Elaboración Propia

Esta clase tiene propiedades en las cuales se debe especificar los nombres de los proyectos en los cuales se va a generar código:

Clase: DomainEntityModel		
Propiedad	**Tipo**	**Descripción**
DomainEntityNamespace	String	Especifica el nombre del proyecto en el que se van a generar las entidades de dominio.
DataBaseImplementationNamespace	String	Especifica el nombre del proyecto en el que se van a generar los scripts de creación de tablas y procedimientos almacenados.
DataAccessNamespace	String	Especifica el nombre del proyecto en el que se van a generar las clases de acceso a datos.
CrossCuttingNamespace	String	Especifica el nombre del proyecto en el que se van a generar las clases comunes y de configración.

Tabla 6.3.1.Propiedades de la clase DomainEntityModel
Fuente: Elaboración Propia

Clase DomainEntity

Esta clase está definida para representar una entidad del dominio, esta es una clase que está contenida en la clase *DomainEntityModel* mediante la relación *DomainEntityModelHasDomainEntities*, que especifica que una Clase *DomainEntityModel* contiene de 0 a muchas clases *DomainEntity*, y una clase *DomainEntity* solo puede tener como contenedora a una sola clase *DomainEntityModel*, como se muestra en la siguiente figura.

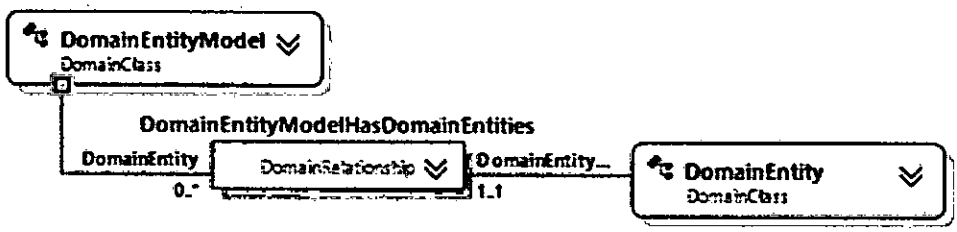


Figura 6.3.3. Relación DomainEntityModelHasDomainEntities
Fuente: Elaboración Propia

La estructura de la clase *DomainEntity* es la siguiente:

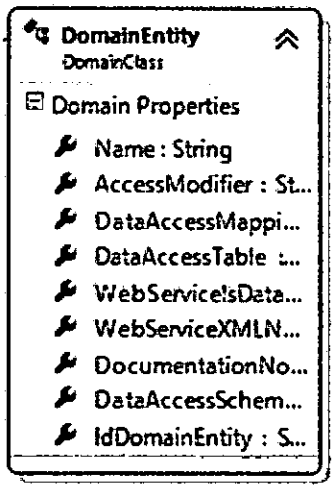


Figura 6.3.4. Clase DomainEntity
Fuente: Elaboración Propia

Las propiedades de la clase *DomainEntity* son las siguientes:

Clase: DomainEntity		
Propiedad	**Tipo**	**Descripción**
Name	String	Nombre de la entidad de dominio.
AccessModifier	String	Modificador de acceso de la entidad de dominio.
DataAccessMapping	Boolean	Especifica si la entidad de dominio tiene una tabla de base de datos asociada.
DataAccessTable	String	Nombre de la tabla de base de datos asociada a la entidad de dominio.
WebServiceIsDataContract	Boolean	Especifica si la entidad de dominio es usada como un contrato de datos para WCF.
WebServiceXMLNamespace	String	Espacio de nombres XML del contrato de datos.
DocumentationNotes	String	Notas para documentación de la entidad de dominio.
DataAccessSchema	String	Especifica el esquema de la tabla de base de datos asociada a la entidad de dominio.
IdDomainEntity	String	Identificador de la entidad de dominio.

Tabla 6.3.2.Propiedades de la clase DomainEntity
Fuente: Elaboración Propia

La clase *DomainEntity* puede estar relacionada con una clase *DomainEntity* mediante una relación de referencia *DomainEntityReferencesTargetDomainEntities* de muchos a muchos como muestra la siguiente figura:

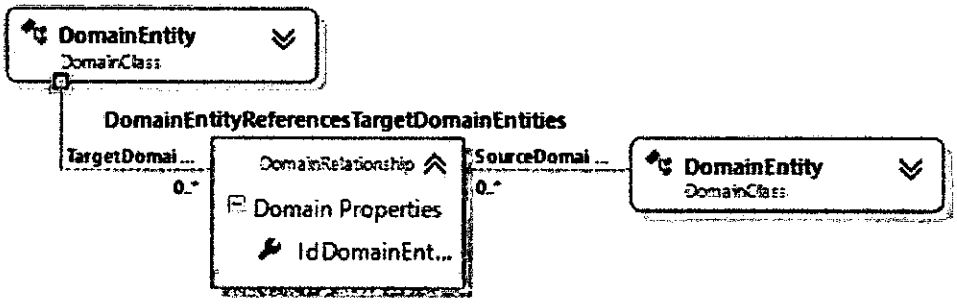


Figura 6.3.5. Relación DomainEntityReferencesTargetDomainEntities
Fuente: Elaboración Propia

Clase PrimitiveProperty

Esta clase representa propiedad primitiva de una entidad del dominio, es decir una propiedad de un tipo nativo como String, Boolean, Int32, etc. esta es una clase que está contenida en la clase *DomainEntity* mediante la relación *DomainEntityHasPrimitiveProperties*, que especifica que una Clase *DomainEntity* contiene de 0 a muchas clases *PrimitiveProperty*, y una clase *PrimitiveProperty* solo puede tener como clase contenedora a una sola clase *DomainEntity*, como se muestra en la siguiente figura.

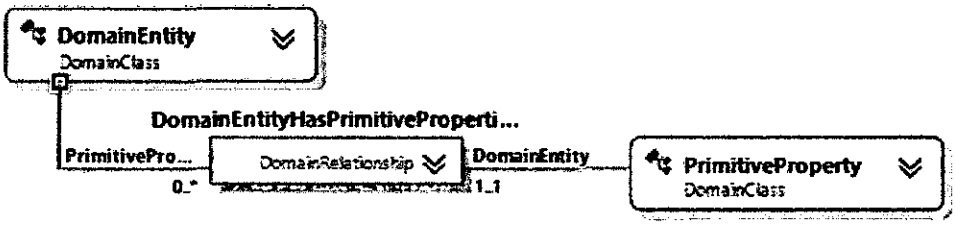


Figura 6.3.6. Relación DomainEntityHasPrimitiveProperties
Fuente: Elaboración Propia

La estructura de la clase *PrimitiveProperty* es la siguiente:

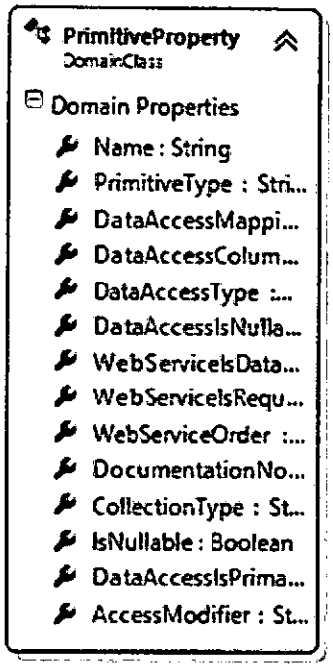


Figura 6.3.7. Clase PrimitiveProperty
Fuente: Elaboración Propia

Y las propiedades de una clase de propiedad primitiva *PrimitiveProperty* son las siguientes:

Clase a PrimitiveProperty		
Propiedad	**Tipo**	**Descripción**
Name	String	Nombre de la propiedad primitiva de la entidad de dominio.
PrimitiveType	String	Tipo base de la propiedad primitiva de la entidad de dominio.
DataAccessMapping	Boolean	Especifica si la propiedad primitiva tiene una columna de tabla de base de datos asociada.
DataAccessColumn	String	Nombre de la columna de tabla de base de datos asociada a la propiedad primitiva de la entidad de dominio.
DataAccessType	String	Tipo de Dato SQL de la columna asociada a la propiedad primitiva de la entidad de dominio.
DataAccessIsNullable	Boolean	Especifica si la columna de base de datos asociada a la propiedad primitiva acepta nulos.
WebServiceIsDataMember	Boolean	Especifica si la propiedad primitiva es miembro de datos del contrato de datos asociado.
WebServiceIsRequired	Boolean	Especifica si el miembro de datos asociado a la propiedad primitiva es obligatorio.
WebServiceOrder	Int32	Especifica el orden único del miembro de datos dentro del contrato de datos asociado.
DocumentationNotes	String	Notas para documentación de la propiedad primitiva asociada a la entidad de dominio.
CollectionType	String	Especifica el tipo de colección de la propiedad primitiva.
IsNullable	Boolean	Especifica si la propiedad primitiva es obligatoria.
DataAccessIsPrimaryKey	Boolean	Especifica si la propiedad primitiva actua como clave primaria de la tabla de base de datos asociada.
AccessModifier	String	Modificador de acceso de la propiedad primitiva.

Tabla 6.3.3.Propiedades de la clase PrimitiveProperty
Fuente: Elaboración Propia

Clase DomainEntityProperty

Esta clase representa propiedad de entidad de dominio contenida en una entidad del dominio, es decir una propiedad de un tipo entidad como Cliente, Producto, Venta, etc. esta es una clase que está contenida en la clase *DomainEntity* mediante la relación *DomainEntityHasDomainEntityProperties*, que especifica que una Clase *DomainEntity* contiene de 0 a muchas clases *DomainEntityProperty*, y una clase *DomainEntityProperty* solo puede tener como clase contenedora a una sola clase *DomainEntity*, como se muestra en la siguiente figura.

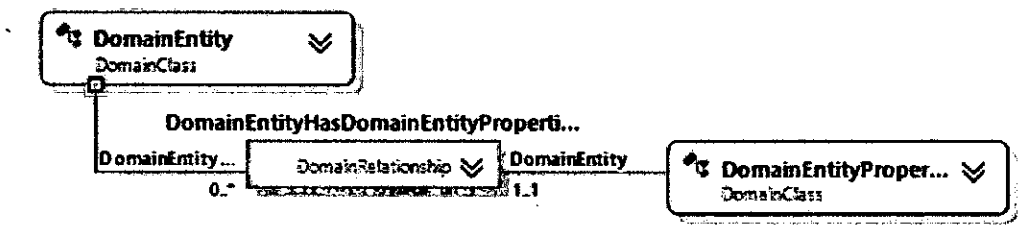


Figura 6.3.8. Relación DomainEntityHasDomainEntityProperties
Fuente: Elaboración Propia

La estructura de la clase *DomainEntityProperty* es la siguiente:

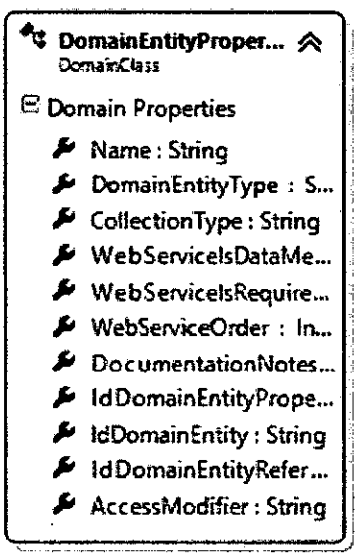


Figura 6.3.9. Clase DomainEntityProperty
Fuente: Elaboración Propia

Y las propiedades de una clase de propiedad primitiva *DomainEntityProperty* son las siguientes:

Clase aDomainEntityProperty		
Propiedad	**Tipo**	**Descripción**
Name	String	Nombre de la propiedad de entidad de dominio.
DomainEntityType	String	Tipo de entidad de dominio de la propiedad de entidad de dominio.
CollectionType	String	Especifica el tipo de colección de la propiedad de entidad de dominio.
WebServiceIsDataMember	Boolean	Especifica si la propiedad de entidad de dominio es miembro de datos del contrato de datos asociado.
WebServiceIsRequired	Boolean	Especifica si el miembro de datos asociado a la propiedad de entidad de dominio es obligatorio.
WebServiceOrder	Int32	Especifica el orden único del miembro de datos dentro del contrato de datos asociado.
DocumentationNotes	String	Notas para documentación de la propiedad de la entidad de dominio asociada a la entidad de dominio.
IdDomainEntityProperty	String	Identificador de la propiedad de entidad de dominio.
IdDomainEntity	String	Identificador de la entidad de dominio a la que pertenece.
IdDomainEntityReferencesTargetDomainEntities	String	Identificador de la relación de referencia de entidad de dominio a entidad de dominio a la que pertenece.
AccessModifier	String	Modificador de acceso de la propiedad de entidad de dominio.

Tabla 6.3.4.Propiedades de la clase DomainEntityProperty
Fuente: Elaboración Propia

Clase DomainEntityCollection

Esta clase está definida para representar una colección de entidades del dominio, esta es una clase que está contenida en la clase *DomainEntityModel* mediante la relación *DomainEntityModelHasDomainEntityCollections*, que especifica que una Clase *DomainEntityModel* contiene de 0 a muchas clases *DomainEntityCollection*, y una clase *DomainEntityCollection* solo puede tener como contenedora a una sola clase *DomainEntityModel*, como se muestra en la siguiente figura.

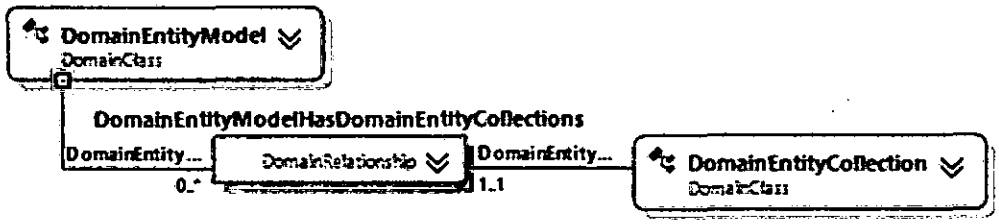


Figura 6.3.10. Relación DomainEntityModelHasDomainEntityCollections
Fuente: Elaboración Propia

La estructura de la clase *DomainEntityCollection* es la siguiente:

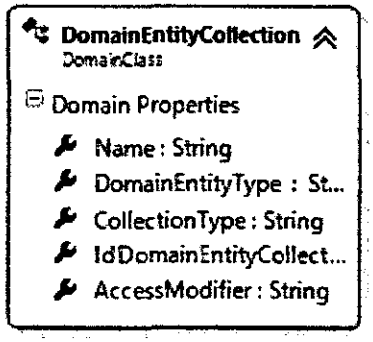


Figura 6.3.11. Clase DomainEntityCollection
Fuente: Elaboración Propia

Las propiedades de la clase *DomainEntityCollection* son las siguientes:

Clase : DomainEntityCollection		
Propiedad	Tipo	Descripción
Name	String	Nombre de la colección de entidades de dominio.
DomainEntityType	String	Nombre de la entidad de dominio asociada a la colección.
CollectionType	Boolean	Tipo de colección de la entidad de dominio.
IdDomainEntityCollection	String	Identificador de la colección de entidades de dominio.
AccessModifier	Boolean	Modificador de acceso de la colección de entidades de dominio.

Tabla 6.3.5.Propiedades de la clase DomainEntity
Fuente: Elaboración Propia

La clase *DomainEntityCollection* puede estar relacionada con una clase *DomainEntity* mediante una relación de referencia *DomainEntityReferencesDomainEntityCollections* que especifica que una colección de entidades tiene una clase *DomainEntity* especificando el tipo de entidad con el que se construye la colección, como se muestra en la siguiente figura:

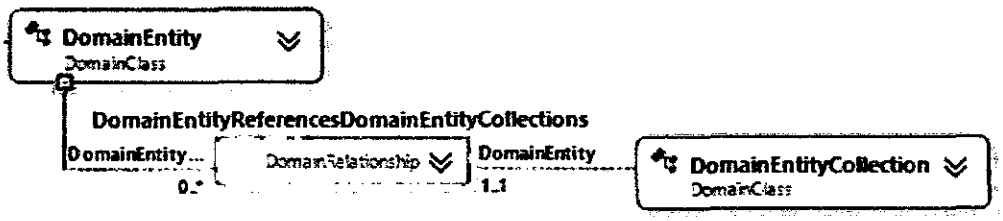


Figura 6.3.12. Relación DomainEntityReferencesDomainEntityCollections
Fuente: Elaboración Propia

Clase NLayerDSLToolsDiagram

Esta clase representa el diagrama de entidades de dominio en el diagrama implementado mediante este modelo, su estructura es la siguiente:

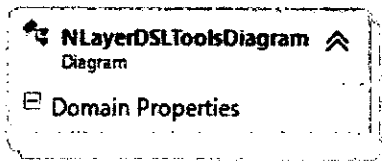


Figura 6.3.13. Clase NLayerDSLToolsDiagram
Fuente: Elaboración Propia

Clase DomainEntityShape

Esta clase representa la forma gráfica de una entidad de dominio en el diagrama de entidades de dominio, está asociado a la clase *DomainEntity* como clase base, su estructura es la siguiente:

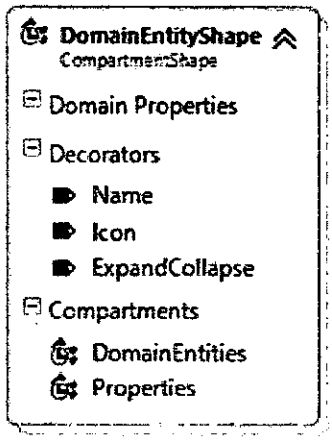


Figura 6.3.14. Clase DomainEntityShape
Fuente: Elaboración Propia

Decoradores: Los decoradores de la forma *DomainEntityShape* son:

Name: Muestra el nombre de la entidad de dominio
Icon: Muestra un icono de entidad de dominio
ExpanCollapse: Muestra un control para expandir o contraer la entidad.

Compartimientos: La forma tiene implementados los siguientes compartimientos:

DomainEntities: Muestra la lista de propiedades entidades de dominio de la entidad.
Properties: Muestra la lista de propiedades primitivas de la entidad.

El ejemplo de una forma (Shape) de entidad de dominio.

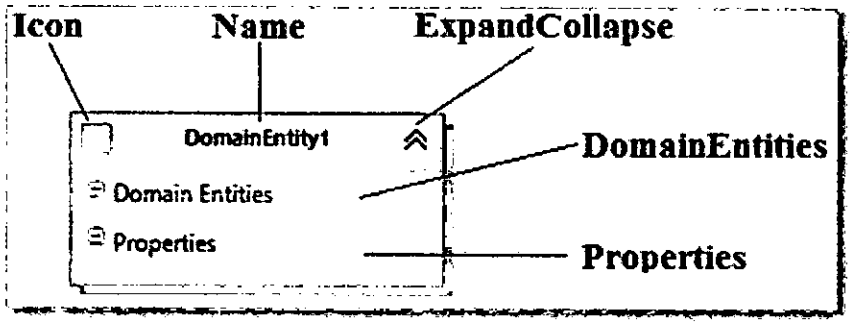


Figura 6.3.15. Ejemplo de un shape de entidad de dominio
Fuente: Elaboración Propia

Clase DomainEntityCollectionShape

Esta clase representa la forma gráfica de una colección de entidades de dominio en el diagrama de entidades de dominio, está asociado a la clase *DomainEntityCollection* como clase base y su estructura es la siguiente:

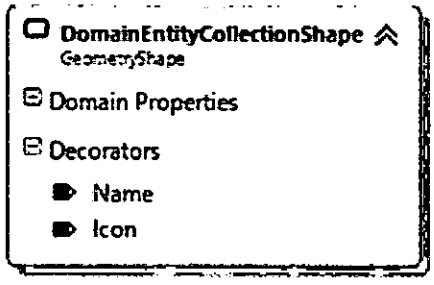


Figura 6.3.16. Clase DomainEntityCollectionShape
Fuente: Elaboración Propia

Decoradores: Los decoradores de la forma *DomainEntityCollectionShape* son:

- **Name:** Muestra el nombre de la colección de entidades de dominio
  Icon: Muestra un icono de la colección de entidades de dominio

El ejemplo de una forma (Shape) de colección de entidades de dominio.

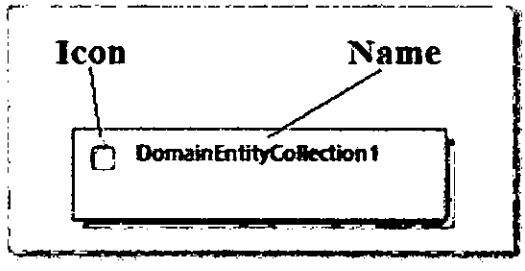


Figura 6.3.17. Ejemplo de un shape de colección de entidades de dominio
Fuente: Elaboración Propia

Clase DomainEntityConnector

Esta clase representa la forma gráfica de una relación de referencia entre una entidad de dominio y una entidad de dominio, está asociado a la clase de referencia *DomainEntityReferencesTargetDomainEntities*, su estructura es la siguiente:

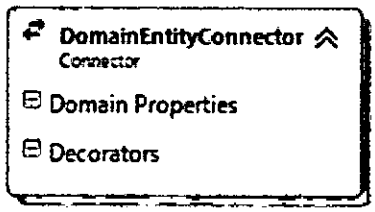


Figura 6.3.18. Clase DomainEntityCollectionShape
Fuente: Elaboración Propia

Clase DomainEntityCollectionConnector

Esta clase representa la forma gráfica de una relación de referencia entre una entidad de dominio y una colección de entidades de dominio, está asociado a la clase de referencia *DomainEntityReferencesDomainEntityCollections*, su estructura es la siguiente:

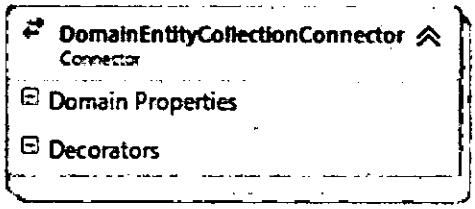


Figura 6.3.19. Clase DomainEntityCollectionConnector
Fuente: Elaboración Propia

Como ejemplo, el diagrama DSL de acuerdo al diseño de entidades quedaría de la siguiente forma:

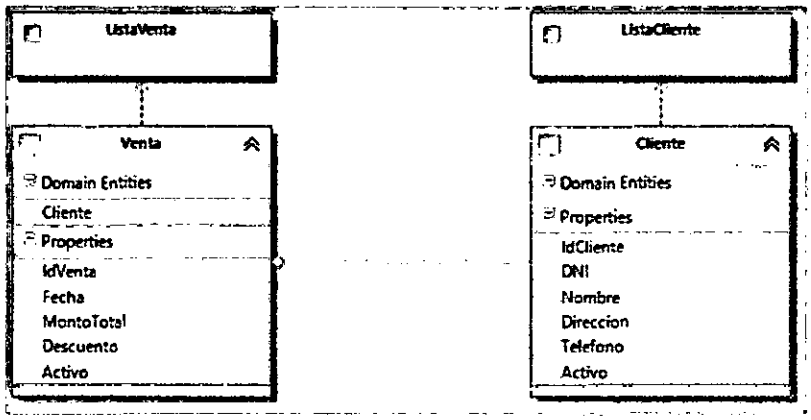


Figura 6.3.20. Ejemplo de diagrama de entidades de dominio
Fuente: Elaboración Propia

Las entidades se generarán tomando en cuenta las propiedades de las clases *DomainEntityModel*, *DomainEntity*, *DomainEntityProperty* y *PrimitiveProperty* para generar el código de cada entidad:

Entidades de dominio: La estructura de los archivos de salida (.cs) tendrán el siguiente formato:

Nombre: [DomainEntity.Name].cs

Contenido:

```
namespace [DomainEntityModel.DomainEntityNamespace]
{
    [DomainEntity.AccessModifier] partial class [DomainEntity.Name]
    {

        public [DomainEntityProperty.DomainEntityType] [DomainEntityType[0].Name] { get; set; }

        ...

        public [DomainEntityProperty.DomainEntityType] [DomainEntityType[N].Name] { get; set; }

        public [PrimitiveProperty.PrimitiveType] [PrimitiveProperty[0].Name] { get; set; }

        ...

        public [PrimitiveProperty.PrimitiveType] [PrimitiveProperty[N].Name] { get; set; }

    }
}
```

Colecciones de entidades de dominio: La estructura de los archivos de salida (.cs) tendrán el siguiente formato:

Nombre: Lista[DomainEntity.Name].cs

Contenido:

```
namespace [DomainEntityModel.DomainEntityNamespace]
{
    [DomainEntity.AccessModifier] partial class Lista[DomainEntity.Name] :
    System.Collections.Generic.[CollectionType]<[DomainEntity.Name]>
    {

    }
}
```

Los objetos de código C# (.cs) se generarán en el proyecto especificado en la propiedad *DomainEntityNamespace* de la entidad de modelo *DomainEntityModel*.

6.4. Diseño del componente de base de datos

El componente de base de datos a generar se basa en la estructura de entidades definida en la sección anterior 6.3 Diseño del componente de entidad. Para ello se utilizarán las clases siguientes:

Clase DomainEntity: Se utilizará la sección Data Access para generar el nombre de la tabla en el script de creación de tabla y los procedimientos almacenados, se usarán las siguientes propiedades.

Propiedad	Descripción
DataAccessMapping	Se genera objetos de datos si esta propiedad está establecida en true.
DataAccessTable	Se extrae esta propiedad para generar el nombre de la tabla de base de datos.
DataAccessSchema	Se extrae esta propiedad para generar el nombre del esquema de la tabla de base de datos.

Tabla 6.4.1. Propiedades de la clase DomainEntity para generar objetos de datos
Fuente: Elaboración Propia

Clase PrimitiveProperty: Se utilizará la sección Data Access para generar el nombre de la columna de la tabla en el script de creación de tablas y para recuperar campos y como parámetros los procedimientos almacenados, se usarán las siguientes propiedades.

Propiedad	Descripción
DataAccessMapping	Se genera objetos de datos si esta propiedad está establecida en true.
DataAccessColumn	Se extrae esta propiedad para generar el nombre de la columna de la tabla de base de datos.
DataAccessType	Se extrae esta propiedad para generar el tipo de dato de la columna de la tabla de base de datos.
DataAccessIsNullable	Se extrae esta propiedad para especificar si la columna de la tabla de base de datos acepta nulos.
DataAccessIsPrimaryKey	Se extrae esta propiedad para especificar si la columna de la tabla de base de datos es clave primaria.

Tabla 6.4.2. Propiedades de la clase PrimitiveProperty para generar objetos de datos
Fuente: Elaboración Propia

El archivo generado contendrá los siguientes objetos de base de datos:

- ✓ Tabla de base de datos.
- ✓ Procedimiento Almacenado de Inserción.
- ✓ Procedimiento Almacenado de Actualización.
- ✓ Procedimiento Almacenado de Eliminación.
- ✓ Procedimiento Almacenado de Consulta Por Id.
- ✓ Procedimiento Almacenado de Consulta Total (Se pueden personalizar filtros).

La estructura del archivo de salida (.sql) tendrá el siguiente formato:

Nombre: Objects_[DataAccessSchema]_[DataAccessTable].sql

Contenido:

```
--SCRIPT DE ELIMINACIÓN Y CREACIÓN DE TABLA
drop table [DataAccessSchema].[DataAccessTable];
go
create table [DataAccessSchema].[DataAccessTable] (
    [DataAccessColumn] [DataAccessType], --PRIMARY KEY
    [DataAccessColumn] [DataAccessType], --FOREIGN KEYS
    [DataAccessColumn] [DataAccessType], --COLUMNS (0)
    ...
);
go

--SCRIPT DE ELIMINACIÓN Y CREACIÓN DEL PROCEDIMIENTO DE INSERCIÓN
drop procedure [DataAccessSchema].[Usp_Ins_[DataAccessTable]];
go
create procedure [DataAccessSchema].[Usp_Ins_[DataAccessTable]]
    @[DataAccessColumn] [DataAccessType], --PRIMARY KEY
    @[DataAccessColumn] [DataAccessType], --FOREIGN KEYS
    @[DataAccessColumn] [DataAccessType], --COLUMNS (0)
    ...
as
begin
    insert into [DataAccessSchema].[DataAccessTable]
    (
        [DataAccessColumn] --PRIMARY KEY
        [DataAccessColumn] --FOREIGN KEYS
        [DataAccessColumn] --COLUMNS (0)
        ...
    )
    values
    (
        @[DataAccessColumn] --PRIMARY KEYS
        @[DataAccessColumn] --FOREIGN KEYS
        @[DataAccessColumn] --COLUMNS (0)
        ...
    );

    SET @[DataAccessColumn] = @@identity; --PRIMARY KEY
end;
go

--SCRIPT DE ELIMINACIÓN Y CREACIÓN DEL PROCEDIMIENTO DE ACTUALIZACIÓN
drop procedure [DataAccessSchema].[Usp_Upd_[DataAccessTable]];
go
create procedure [DataAccessSchema].[Usp_Upd_[DataAccessTable]]
    @[DataAccessColumn] [DataAccessType], --PRIMARY KEY
    @[DataAccessColumn] [DataAccessType], --FOREIGN KEYS
    @[DataAccessColumn] [DataAccessType], --COLUMNS (0)
    ...
as
begin
    update [DataAccessSchema].[DataAccessTable] set
        [DataAccessColumn] = @[DataAccessColumn], --PRIMARY KEY
        [DataAccessColumn] = @[DataAccessColumn], --FOREIGN KEYS
        [DataAccessColumn] = @[DataAccessColumn], --COLUMNS (0)
        ...
    where
        [DataAccessColumn] = @[DataAccessColumn]; --PRIMARY KEY
end;
go
```

```
--SCRIPT DE ELIMINACIÓN Y CREACIÓN DEL PROCEDIMIENTO DE ELIMINACIÓN
drop procedure [DataAccessSchema].[Usp_Del_[DataAccessTable]];
go
create procedure [DataAccessSchema].[Usp_Del_[DataAccessTable]]
    @[DataAccessColumn] [DataAccessType], --PRIMARY KEY
as
begin
    delete from [DataAccessSchema].[DataAccessTable]
    where
        [DataAccessColumn] = @[DataAccessColumn]; --PRIMARY KEY
end;
go

--SCRIPT DE ELIMINACIÓN Y CREACIÓN DEL PROCEDIMIENTO DE SELECCIÓN POR ID
drop procedure [DataAccessSchema].[Usp_Sel_[DataAccessTable]_PorId];
go
create procedure [DataAccessSchema].[Usp_Sel_[DataAccessTable]_PorId]
    @[DataAccessColumn] [DataAccessType], --PRIMARY KEY
as
begin
    select
        [DataAccessColumn], --PRIMARY KEY
        [DataAccessColumn], --FOREIGN KEYS
        [DataAccessColumn], --COLUMNS (0)
        ...
    from [DataAccessSchema].[DataAccessTable]
    where
        [DataAccessColumn] = @[DataAccessColumn]; --PRIMARY KEY
end;
go

--SCRIPT DE ELIMINACIÓN Y CREACIÓN DEL PROCEDIMIENTO DE SELECCIÓN
drop procedure [DataAccessSchema].[Usp_Sel_[DataAccessTable]];
go
create procedure [DataAccessSchema].[Usp_Sel_[DataAccessTable]]
    @[DataAccessColumn] [DataAccessType], --PRIMARY KEY
    @[DataAccessColumn] [DataAccessType], --FOREIGN KEYS
    @[DataAccessColumn] [DataAccessType], --COLUMNS (0)
    ...
as
begin
    select
        [DataAccessColumn], --PRIMARY KEY
        [DataAccessColumn], --FOREIGN KEYS
        [DataAccessColumn], --COLUMNS (0)
        ...
    from [DataAccessSchema].[DataAccessTable]
    --where (poner tu filtro aquí)
end;
go
```

Los objetos de script (.sql) se generarán en el proyecto especificado en la propiedad *DataBaseImplementationNamespace* de la entidad de modelo *DomainEntityModel*.

6.5. Diseño del componente de acceso a datos

El componente de acceso a datos también se va a generar de acuerdo al modelo de entidades de dominio especificado en el proyecto de Modelado y Diseño de la solución del proyecto para ello se utilizarán las clases *DomainEntityModel*, *DomainEntity*, *DomainEntityProperty* y *PrimitiveProperty* para generar el código de acceso a datos de cada entidad.

La estructura de los archivos de salida (.cs) tendrán el siguiente formato:

La cabecera de cada clase de acceso a datos es la siguiente:

```
using System;
using System.Data;
using System.Data.Common;
using Microsoft.Practices.EnterpriseLibrary.Data;
using [DomainEntityModel.CrossCuttingNamespace];
using [DomainEntityModel.DomainEntityNamespace];

namespace [DomainEntityModel.DataAccessNamespace]
{
    public partial class [DomainEntity.Name]DataAccess
    {
        private static DatabaseProviderFactory oDatabaseProviderFactory = new
        DatabaseProviderFactory();
        private Database oDatabase =
        oDatabaseProviderFactory.Create(Conexion.cnConexion);

        ...
    }
}
```

Se generarán los siguientes métodos de acceso a datos:

Registrar entidad de dominio

```
//********************** FUNCION PARA REGISTRO DE [DomainEntity.Name] **********************//
public [DomainEntity.Name] Registrar_[DomainEntity.Name]([DomainEntity.Name] [domainEntity.Name])
{
        DbCommand oDbCommand = oDatabase.GetStoredProcCommand(Procedimiento.USP_INS_[DomainEntity.Name]);

        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Primary Key
        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Foreign Key
         ...
        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //All Other Columns

        oDatabase.ExecuteNonQuery(oDbCommand);

        [domainEntity.Name.DomainEntityProperty.Name] = //Primary Key
        DataUtil.DbValueToDefault<System.[DomainEntityProperty.PrimitiveType]>
        (oDatabase.GetParameterValue(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]"));

        return [domainEntity.Name];
}
```

Actualizar entidad de dominio

```
//********************** FUNCION PARA ACTUALIZACION DE [DomainEntity.Name] **********************//
public [DomainEntity.Name] Actualizar_[DomainEntity.Name]([DomainEntity.Name] [domainEntity.Name])
{
        DbCommand oDbCommand = oDatabase.GetStoredProcCommand(Procedimiento.USP_UPD_[DomainEntity.Name]);

        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Primary Key
        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Foreign Key
         ...
        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //All Other Columns

        oDatabase.ExecuteNonQuery(oDbCommand);

        return [domainEntity.Name];
}
```

Eliminar entidad de dominio

```
//********************** FUNCION PARA ELIMINACION DE [DomainEntity.Name] **********************//
public [DomainEntity.Name] Eliminar_[DomainEntity.Name](
  [domainEntity.Name.DomainEntityProperty.PrimitiveType] [domainEntity.Name.DomainEntityProperty.Name]
) //Primary Key
{
        DbCommand oDbCommand = oDatabase.GetStoredProcCommand(Procedimiento.USP_DEL_[DomainEntity.Name]);

        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Primary Key

        oDatabase.ExecuteNonQuery(oDbCommand);

        return [domainEntity.Name];
}
```

Obtener entidad de dominio por id

```
//********************** FUNCION PARA SELECCION POR ID DE [DomainEntity.Name] **********************//
public [DomainEntity.Name] Obtener_[DomainEntity.Name] _PorId(
  [domainEntity.Name.DomainEntityProperty.PrimitiveType] [domainEntity.Name.DomainEntityProperty.Name]
) //Primary Key
{
        [DomainEntity.Name] [domainEntity.Name] = new [DomainEntity.Name]();
        DbCommand oDbCommand = oDatabase.GetStoredProcCommand(
        Procedimiento.USP_SEL_[DomainEntity.Name]_POR_ID);

        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Primary Key

        using (IDataReader oIDataReader = oDatabase.ExecuteReader(oDbCommand))
        {
          while (oIDataReader.Read())
          {
            [domainEntity.Name.DomainEntityProperty.Name] =
            DataUtil.DbValueToDefault<System. [DomainEntityProperty.PrimitiveType]>
            (oIDataReader[oIDataReader.GetOrdinal("[domainEntity.Name.DomainEntityProperty.Name]")]);
            ... //Retrieve All Columns
            [domainEntity.Name.DomainEntityProperty.Name] =
            DataUtil.DbValueToDefault<System. [DomainEntityProperty.PrimitiveType]>
            (oIDataReader[oIDataReader.GetOrdinal("[domainEntity.Name.DomainEntityProperty.Name]")]);
          }
        }

        return [domainEntity.Name];
}
```

Listar entidad de dominio

```
//********************** FUNCION PARA SELECCION DE [DomainEntity.Name] **********************//
public [DomainEntity.Name] Listar_[DomainEntity.Name]([DomainEntity.Name] [domainEntity.Name])
{
        Lista[DomainEntity.Name] lista[domainEntity.Name] = new Lista[DomainEntity.Name]();
        DbCommand oDbCommand = oDatabase.GetStoredProcCommand(Procedimiento.USP_SEL_[DomainEntity.Name]);

        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Primary Key
        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //Foreign Key
         ...
        oDatabase.AddOutParameter(oDbCommand, "@[domainEntity.Name.DomainEntityProperty.Name]",
          DbType.[DomainEntityProperty.PrimitiveType], [domainEntity.Name.DomainEntityProperty.Name]
        ); //All Other Columns

        using (IDataReader oIDataReader = oDatabase.ExecuteReader(oDbCommand))
        {
          while (oIDataReader.Read())
          {
            [domainEntity.Name.DomainEntityProperty.Name] =
            DataUtil.DbValueToDefault<System. [DomainEntityProperty.PrimitiveType]>
            (oIDataReader[oIDataReader.GetOrdinal("[domainEntity.Name.DomainEntityProperty.Name]")]);
            ... //Retrieve All Columns
            [domainEntity.Name.DomainEntityProperty.Name] =
            DataUtil.DbValueToDefault<System. [DomainEntityProperty.PrimitiveType]>
            (oIDataReader[oIDataReader.GetOrdinal("[domainEntity.Name.DomainEntityProperty.Name]")]);
            lista[domainEntity.Name].Add([domainEntity.Name]);
          }
        }

        return lista[domainEntity.Name];
}
```

Los objetos de código C# (.sql) de acceso a datos se generarán en el proyecto especificado en la propiedad *DataAccessNamespace* de la entidad de modelo *DomainEntityModel*.

Adicionalmente se crean clases comunes para recuperar los nombres de la conexión y los nombres de los procedimientos almacenados:

Clase Conexion.cs

```
using System;

namespace DomainEntityModel.CrossCuttingNamespace
{
    public class Conexion
    {
        public const String cnConexion = "cnConexion";
    }
}
```

Clase Procedimiento.cs

```
using System;

namespace DomainEntityModel.CrossCuttingNamespace
{
  public class Procedimiento
  {

    #region PROCEDIMIENTOS ALMACENADOS [DomainEntity.Name]
    public const String USP_INS_[DomainEntity.Name] = "[dbo].[Usp_Ins_[DomainEntity.Name]]";
    public const String USP_UPD_[DomainEntity.Name] = "[dbo].[Usp_Upd_[DomainEntity.Name]]";
    public const String USP_DEL_[DomainEntity.Name] = "[dbo].[Usp_Del_[DomainEntity.Name]]";
    public const String USP_SEL_[DomainEntity.Name]_POR_ID = "[dbo].[Usp_Sel_[DomainEntity.Name]_PorId]";
    public const String USP_SEL_[DomainEntity.Name] = "[dbo].[Usp_Sel_[DomainEntity.Name]]";
    #endregion
    ...
    //Store procedure constants for all domain entities
  }
}
```

Los objetos de código C# (.sql) de estas clases utilitarias se generarán en el proyecto especificado en la propiedad *CrossCuttingNamespace* de la entidad de modelo *DomainEntityModel*.

CAPÍTULO VII: DESARROLLO DE LA SOLUCIÓN PROPUESTA

7.1. Desarrollo de generación la plantilla de la solución

El desarrollo de la plantilla de solución se encuentra en el repositorio del control de código fuente del proyecto:

Dirección web del repositorio de código fuente: Se puede acceder al código fuente del proyecto de Visual Studio en el siguiente enlace:

https://devframework.visualstudio.com/DefaultCollection/DevFramework/_versionControl

Fuentes la de plantilla de Visual Studio

Solución: N-Layer-DSL-Tools

Proyecto	Archivo
N-Layer-DSL-Tools-Template	N-Layer-DSL-Tools-Template.vstemplate
N-Layer-DSL-Tools-Template	N-Layer-DSL-Tools-Template.ico
N-Layer-DSL-Tools-Template	Application.Core/Application.Core.vstemplate
N-Layer-DSL-Tools-Template	Application.Core/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Application.Core/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Application.Core/ Template.ico
N-Layer-DSL-Tools-Template	Application.MainModule/Application.MainModule.vstemplate
N-Layer-DSL-Tools-Template	Application.MainModule/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Application.MainModule/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Application.MainModule/Template.ico
N-Layer-DSL-Tools-Template	CrossCutting.Common/CrossCutting.Common.vstemplate
N-Layer-DSL-Tools-Template	CrossCutting.Common/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	CrossCutting.Common/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	CrossCutting.Common/Template.ico
N-Layer-DSL-Tools-Template	Data.Core/Data.Core.vstemplate
N-Layer-DSL-Tools-Template	Data.Core/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Data.Core/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Data.Core/Template.ico
N-Layer-DSL-Tools-Template	Data.MainModule/Data.MainModule.vstemplate
N-Layer-DSL-Tools-Template	Data.MainModule/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Data.MainModule/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Data.MainModule/Template.ico
N-Layer-DSL-Tools-Template	DataBase.Implementation/DataBase.Implementation.vstemplate
N-Layer-DSL-Tools-Template	DataBase.Implementation/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	DataBase.Implementation/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	DataBase.Implementation/Template.ico
N-Layer-DSL-Tools-Template	DistributedServices.Core/DistributedServices.Core.vstemplate
N-Layer-DSL-Tools-Template	DistributedServices.Core/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	DistributedServices.Core/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	DistributedServices.Core/Template.ico
N-Layer-DSL-Tools-Template	DistributedServices.MainModule/DistributedServices.MainModule.vstemplate
N-Layer-DSL-Tools-Template	DistributedServices.MainModule/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	DistributedServices.MainModule/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	DistributedServices.MainModule/Template.ico
N-Layer-DSL-Tools-Template	Domain.Core/Domain.Core.vstemplate

N-Layer-DSL-Tools-Template	Domain.Core/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Domain.Core/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Domain.Core/Template.ico
N-Layer-DSL-Tools-Template	Domain.Core.Entities/Domain.Core.Entities.vstemplate
N-Layer-DSL-Tools-Template	Domain.Core.Entities/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Domain.Core.Entities/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Domain.Core.Entities/Template.ico
N-Layer-DSL-Tools-Template	Domain.MainModule/Domain.MainModule.vstemplate
N-Layer-DSL-Tools-Template	Domain.MainModule/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Domain.MainModule/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Domain.MainModule/Template.ico
N-Layer-DSL-Tools-Template	Domain.MainModule.Entities/Domain.MainModule.Entities.vstemplate
N-Layer-DSL-Tools-Template	Domain.MainModule.Entities/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Domain.MainModule.Entities/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Domain.MainModule.Entities/Template.ico
N-Layer-DSL-Tools-Template	Modeling.Model/Modeling.Model.vstemplate
N-Layer-DSL-Tools-Template	Modeling.Model/AssemblyInfo.cs
N-Layer-DSL-Tools-Template	Modeling.Model/ProjectTemplate.csproj
N-Layer-DSL-Tools-Template	Modeling.Model/Template.ico
N-Layer-DSL-Tools-Wizard	Resources/SolutionTemplate.resx
N-Layer-DSL-Tools-Wizard	N-Layer-DSL-Tools-Wizard-Key.pfx
N-Layer-DSL-Tools-Wizard	NLayerDSLToolsWizard.cs

Tabla 7.1.1. Fuentes la de plantilla de Visual Studio
Fuente: Elaboración Propia

7.2. Desarrollo del componente de base de datos

El desarrollo del componente de base de datos también se encuentra en el repositorio del control de código fuente del proyecto, las fuentes abarcan dos proyectos *Dsl* y *DslPackage*, para el generar el código del componente de base de datos se usa una plantilla de transformación t4 (*DatabaseCodeGenerator.tt)* que se encuentra en el proyecto *DslPackage* como se subraya en las fuentes

Dirección web del repositorio de código fuente: Se puede acceder al código fuente del proyecto de Visual Studio en el siguiente enlace:

https://devframework.visualstudio.com/DefaultCollection/DevFramework/_versionControl

Fuentes del componente de base de datos

Solución: N-Layer-DSL-Tools

Proyecto	Archivo
Dsl	CodeAnalysisDictionary.xml
Dsl	DslDefinition.dsl
Dsl	GlobalSuppressions.cs
Dsl	CustomCode/Common/DomainEntityModelCodeGeneratorBase.cs
Dsl	CustomCode/Domain Classes/NLayerDSLProjectDomainModel.cs
Dsl	CustomCode/Domain Types/Base/BaseTypeConverter.cs
Dsl	CustomCode/Domain Types/AccessModifierTypeConverter.cs
Dsl	CustomCode/Domain Types/CollectionTypeTypeConverter.cs

Dsl	CustomCode/Domain Types/DomainEntityCollectionTypeTypeConverter.cs
Dsl	CustomCode/Domain Types/DomainEntityTypeConverter.cs
Dsl	CustomCode/Domain Types/PrimitiveTypeTypeConverter.cs
Dsl	CustomCode/Domain Types/ProjectImplementationTypeConverter
Dsl	CustomRules/Domain Classes/DomainEntityCollectionRules.cs
Dsl	CustomRules/Domain Classes/DomainEntityPropertyRules.cs
Dsl	CustomRules/Domain Classes/DomainEntityRules.cs
Dsl	CustomRules/Domain Classes/PrimitivePropertyRules.cs
Dsl	CustomRules/Domain Relationships/DomainEntityReferencesDomainEntityCollectionsRules.cs
Dsl	CustomRules/Domain Relationships/DomainEntityReferencesTargetDomainEntitiesRules.cs
Dsl	GeneratedCode/ConnectionBuilders.tt
Dsl	GeneratedCode/Connectors.tt
Dsl	GeneratedCode/Diagram.tt
Dsl	GeneratedCode/DirectiveProcessor.tt
Dsl	GeneratedCode/DomainClasses.tt
Dsl	GeneratedCode/DomainModel.tt
Dsl	GeneratedCode/DomainModelResx.tt
Dsl	GeneratedCode/DomainRelationships.tt
Dsl	GeneratedCode/HelpKeywordHelper.tt
Dsl	GeneratedCode/MultiplicityValidation.tt
Dsl	GeneratedCode/NLayerDSLToolsSchema.tt
Dsl	GeneratedCode/PropertiesGrid.tt
Dsl	GeneratedCode/SerializationHelper.tt
Dsl	GeneratedCode/Serializer.tt
Dsl	GeneratedCode/Shapes.tt
Dsl	GeneratedCode/ToolboxHelper.tt
Dsl	Resources/Images/DomainEntity.bmp
Dsl	Resources/Images/DomainEntityCollection.bmp
Dsl	Resources/Images/DomainEntityCollectionConnector.bmp
Dsl	Resources/Images/DomainEntityConnector.bmp
Dsl	Resources/ConnectorSourceSearch.cur
Dsl	Resources/ConnectorTargetSearch.cur
Dsl	Resources/ExampleConnectorToolBitmap.bmp
Dsl	Resources/ExampleShapeToolBitmap.bmp
Dsl	Resources/TypeConverters.resx
Dsl	Resources/TypeConverters.Designer.cs
Dsl	Util/GlobalConstants.cs
Dsl	Util/GlobalFunctions.cs
Dsl	Util/GlobalVariables.cs
DslPackage	CustomCode/CommandSet.cs
DslPackage	GeneratedCode/CommandSet.tt
DslPackage	GeneratedCode/Constants.tt
DslPackage	GeneratedCode/DocData.tt
DslPackage	GeneratedCode/DocView.tt
DslPackage	GeneratedCode/EditorFactory.tt
DslPackage	GeneratedCode/GeneratedVSCT.tt
DslPackage	GeneratedCode/ModelExplorer.tt
DslPackage	GeneratedCode/ModelExplorerToolWindow.tt
DslPackage	GeneratedCode/Package.tt
DslPackage	ProjectItemTemplates/CSharp.tt
DslPackage	ProjectItemTemplates/domainentity.diagram
DslPackage	ProjectItemTemplates/domainentity.domainentity

DslPackage	ProjectTemplates/CSharp/1033/N-Layer-DSL-Tools-Template.zip
DslPackage	Resources/File.ico
DslPackage	Resources/ModelExplorerToolWindowBitmaps.bmp
DslPackage	TextTemplates/CommonCodeGenerator.tt
DslPackage	TextTemplates/DataAccessCodeGenerator.tt
DslPackage	TextTemplates/DatabaseCodeGenerator.tt
DslPackage	TextTemplates/EntityCodeGenerator.tt
DslPackage	TextTemplates/EntityCollectionCodeGenerator.tt
DslPackage	Commands.vsct
DslPackage	Key.snk
DslPackage	source.extension.tt
DslPackage	VSPackage.resx

Tabla 7.2.1. Fuentes del componente de base de datos
Fuente: Elaboración Propia

7.3. Desarrollo del componente de entidad

Las fuentes del componente de entidad son las mismas fuentes del componente de base de datos pero para generar tanto el componente de entidad de dominio como el componente de lista de entidades de dominio se usan las plantillas de transformación t4 *EntityCodeGenerator.tt* y *EntityCollectionCodeGenerator.tt* pertenecientes al proyecto *DSLPackage*.

Dirección web del repositorio de código fuente: Se puede acceder al código fuente del proyecto de Visual Studio en el siguiente enlace:

https://devframework.visualstudio.com/DefaultCollection/DevFramework/_versionControl

Fuentes del componente de entidad

Solución: N-Layer-DSL-Tools

Proyecto	Archivo
DslPackage	TextTemplates/EntityCodeGenerator.tt
DslPackage	TextTemplates/EntityCollectionCodeGenerator.tt

Tabla 7.3.1. Fuentes del componente de entidad
Fuente: Elaboración Propia

7.4. Desarrollo del componente de acceso a datos

Las fuentes del componente de acceso a datos también son las mismas fuentes del componente de base de datos pero para generar el componente de acceso a datos se usan las plantillas t4 *CommonCodeGenerator.tt* y *DataAccessCodeGenerator.tt* que pertenecen al proyecto *DSLPackage*.

Dirección web del repositorio de código fuente: Se puede acceder al código fuente del proyecto de Visual Studio en el siguiente enlace:

https://devframework.visualstudio.com/DefaultCollection/DevFramework/_versionControl

Fuentes de componente de acceso a datos

Solución: N-Layer-DSL-Tools

Proyecto	Archivo
DslPackage	TextTemplates/CommonCodeGenerator.tt
DslPackage	TextTemplates/DataAccessCodeGenerator.tt

Tabla 7.4.1. Fuentes del componente de acceso a datos
Fuente: Elaboración Propia

7.5. Documentación de la Arquitectura

Todo lo referente a la arquitectura del proyecto, que está basada en la *Guía de Arquitectura de N-Capas orientada al dominio* de Microsoft se puede consultar en el Capítulo VI del presente documento.

Para una mejor referencia de la arquitectura N-Capas orientada al dominio se puede consultar la guía oficial de Microsoft citada en la Bibliografía del presente documento.

7.6. Manual de Usuario

El manual de usuario de la solución DSL implementada en el proyecto de investigación se puede consultar en el *Anexo A* de la sección de anexos del presente documento del proyecto.

CAPÍTULO VIII: PRUEBAS DE LA SOLUCIÓN PROPUESTA

8.1. Pruebas de la generación de la plantilla del proyecto

Set de Pruebas

Nro	Tipo de Prueba	Escenario	Descripción
01	Funcional	E01	Prueba de generación de plantilla de proyecto.
02	Técnica	E01	Prueba de elaboración de plantilla sin framework.
03	Técnica	E01	Prueba de elaboración de plantilla con framework.

Tabla 8.1.1. Set de pruebas de la generación de la plantilla del proyecto
Fuente: Elaboración Propia

✓ **Prueba 01 – Escenario 01 (01-E01):** Prueba de generación de plantilla de proyecto.

Usuarios:

Usuario	Descripción
Arquitecto de Software	Arquitecto de software encargado de elaborar el framework.
Analista Programador	Analista programador encargado de desarrollar el proyecto.

Tabla 8.1.2. Usuarios de la generación de la plantilla del proyecto
Fuente: Elaboración Propia

Pasos:

Pasos - 01-E01			
Nro	Responsable	Descripción	Resultado
01	Usuario	El usuario ingresa a la herramienta Visual Studio 2012	Ok
02	Usuario	El usuario selecciona el Menú File > New > Project	Ok
03	Usuario	El usuario selecciona la sección Templates > C#	Ok
04	Usuario	El usuario selecciona la plantilla "N-Layered Domain-Oriented Architecture Project"	Ok
05	Usuario	El usuario ingresa el nombre del proyecto "Enterprise.Module"	Ok
06	Usuario	El usuario selecciona la ubicación del proyecto "D:\"	Ok
07	Usuario	El usuario presiona el botón Ok	Ok
08	IDE	El IDE Genera la plantilla del proyecto de N-Layer	Ok

Tabla 8.1.3. Pasos de la generación de la plantilla del proyecto
Fuente: Elaboración Propia

Resultados:

TipoResultado	Descripción	Ejecuciones
Resultado Ok	Se genera la plantilla del proyecto de N-Capas correctamente de acuerdo al diseño especificado en la sección 6.2 del documento.	1
Resultado Error	No se genera la plantilla del proyecto de N-Capas en su totalidad o se genera parcialmente.	0

Tabla 8.1.4. Resultados de la generación de la plantilla del proyecto
Fuente: Elaboración Propia

✓ **Prueba 02 – Escenario 01 (02-E01):** Prueba de elaboración de plantilla sin framework.

Se elaboró la plantilla del framework partiendo desde cero y agregando cada uno de los proyectos manualmente, estos fueron los resultados:

Paso	Descripción	Tiempo
01	Creación de la solución.	1 min
02	Creación de la estructura de carpetas de la solución.	8 min
03	Creación del proyecto Enterprise.Module.Modeling.Model. (No es necesario)	0 min
04	Creación del proyecto Enterprise.Module.DistributedServices.Core.	1 min
05	Creación del proyecto Enterprise.Module.DistributedServices.MainModule.	1 min
06	Creación del proyecto Enterprise.Module.Application.Core.	1 min
07	Creación del proyecto Enterprise.Module.Application.MainModule.	1 min
08	Creación del proyecto Enterprise.Module.Domain.Core.	1 min
09	Creación del proyecto Enterprise.Module.Domain.Core.Entities.	1 min
10	Creación del proyecto Enterprise.Module.Domain.MainModule.	1 min
11	Creación del proyecto Enterprise.Module.Domain.MainModule.Entities.	1 min
12	Creación del proyecto Enterprise.Module.Data.Core.	1 min
13	Creación del proyecto Enterprise.Module.Data.MainModule.	1 min
14	Creación del proyecto Enterprise.Module.CrossCutting.Common.	1 min
15	Creación del proyecto Enterprise.Module.DataBase.Implementation.	1 min
Tiempo Total		21 min

Tabla 8.1.5. Resultados de la generación de la plantilla del proyecto - 02-E01
Fuente: Elaboración Propia

✓ **Prueba 03 – Escenario 01 (03-E01):** Prueba de elaboración de plantilla con framework.

Se elaboró la plantilla del framework haciendo uso de la plantilla del proyecto, estos fueron los resultados:

Paso	Descripción	Tiempo
01	Creación de la solución.	1 min
02	Creación de la estructura de carpetas de la solución.	0.5 seg
03	Creación del proyecto Enterprise.Module.Modeling.Model.	1 seg
04	Creación del proyecto Enterprise.Module.DistributedServices.Core.	0.5 seg
05	Creación del proyecto Enterprise.Module.DistributedServices.MainModule.	0.5 seg
06	Creación del proyecto Enterprise.Module.Application.Core.	0.5 seg
07	Creación del proyecto Enterprise.Module.Application.MainModule.	0.5 seg
08	Creación del proyecto Enterprise.Module.Domain.Core.	0.5 seg
09	Creación del proyecto Enterprise.Module.Domain.Core.Entities.	0.5 seg
10	Creación del proyecto Enterprise.Module.Domain.MainModule.	0.5 seg
11	Creación del proyecto Enterprise.Module.Domain.MainModule.Entities.	0.5 seg
12	Creación del proyecto Enterprise.Module.Data.Core.	0.5 seg
13	Creación del proyecto Enterprise.Module.Data.MainModule.	1 seg
14	Creación del proyecto Enterprise.Module.CrossCutting.Common.	0.5 seg
15	Creación del proyecto Enterprise.Module.DataBase.Implementation.	1 seg
Tiempo Total		1 min, 8.5 seg

Tabla 8.1.6. Resultados de la generación de la plantilla del proyecto - 03-E01
Fuente: Elaboración Propia

Comparación de tiempos: Se han comparado los tiempos de desarrollo de las dos pruebas técnicas:

Nro	Escenario	Descripción	Tiempo (seg)
2	02-E01	Prueba de elaboración de plantilla sin framework.	1260
3	03-E01	Prueba de elaboración de plantilla con framework.	68.5

Tabla 8.1.7. Comparación de tiempos de la generación de la plantilla del proyecto
Fuente: Elaboración Propia

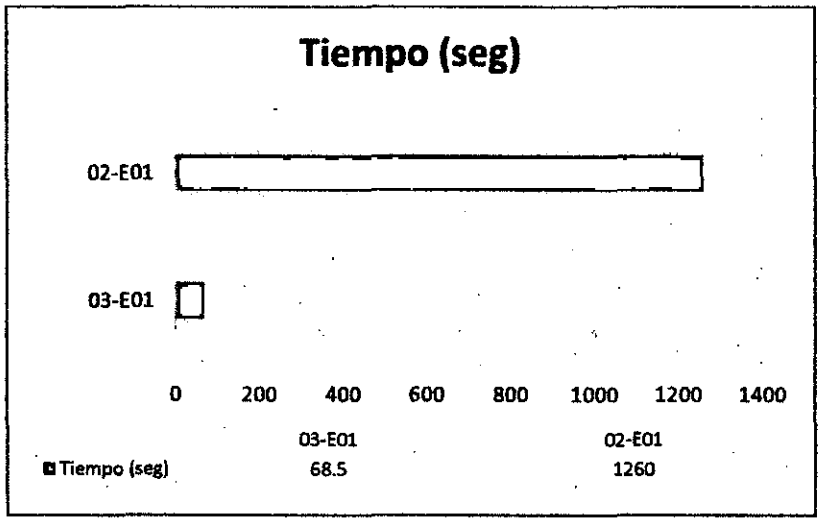


Figura 8.1.1. Comparación de tiempos de la generación de la plantilla del proyecto
Fuente: Elaboración Propia

8.2. Pruebas de la generación del componente de entidad

Set de Pruebas

Nro	Tipo de Prueba	Escenario	Descripción
01	Funcional	E01	Prueba de diseño de modelo de entidades – sección de definición de entidades.
02	Técnica	E01	Prueba de elaboración de entidades sin framework.
03	Técnica	E01	Prueba de elaboración de entidades con framework.

Tabla 8.2.1. Set de pruebas de la generación del componente de entidad
Fuente: Elaboración Propia

- ✓ **Prueba 01 – Escenario 01 (01-E01):** Prueba de diseño de modelo de entidades – sección de definición de entidades.

Usuarios:

Usuario	Descripción
Analista Programador	Analista programador encargado de desarrollar el proyecto.

Tabla 8.2.2. Usuarios de la generación del componente de entidad
Fuente: Elaboración Propia

Formato de Prueba: El usuario tendrá que elaborar el siguiente modelo de entidades de dominio.

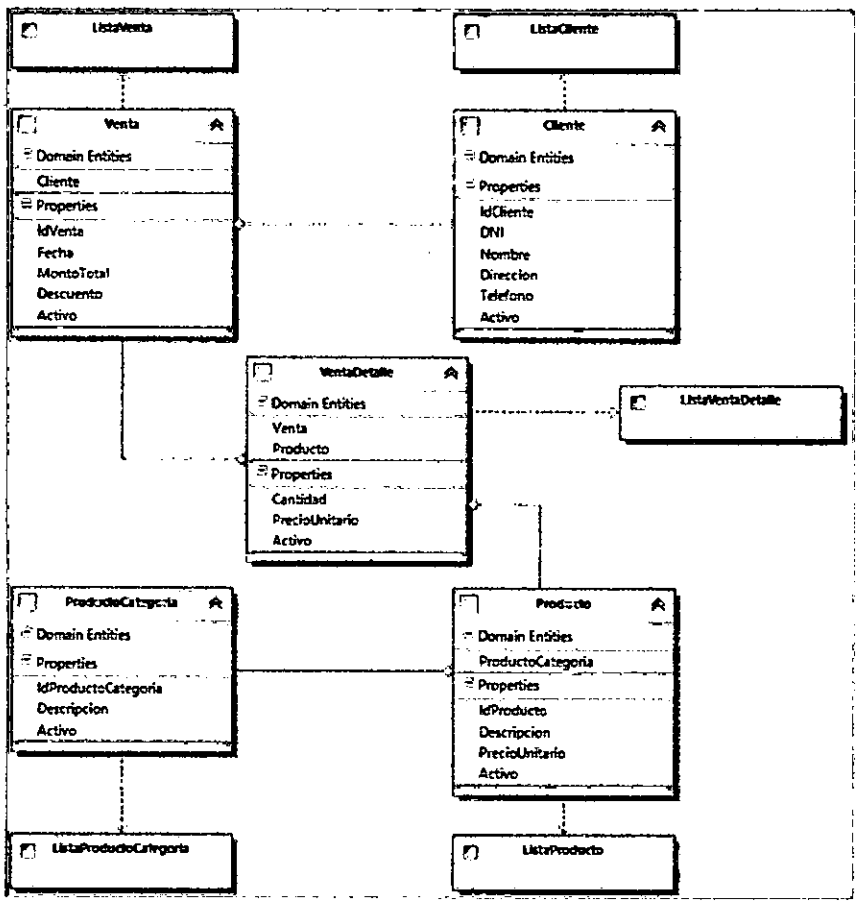


Figura 8.2.1. Modelo de entidades de dominio para prueba
Fuente: Elaboración Propia

Pasos:

Pasos 8.01-E01			
Nro	Responsable	Descripción	Resultado
01	Usuario	El usuario abre el archivo domainentitymodel.domainentity del proyecto de modelado de la solución elaborada en el punto 8.1 del documento.	Ok
02	Usuario	El usuario diseña la entidad Cliente	Ok
03	Usuario	El usuario diseña la entidad Venta	Ok
04	Usuario	El usuario diseña la entidad ProductoCategoria	Ok
05	Usuario	El usuario diseña la entidad Producto	Ok
06	Usuario	El usuario diseña la entidad VentaDetalle	Ok
07	Usuario	El usuario diseña la colección de entidades ListaCliente	Ok
08	Usuario	El usuario diseña la colección de entidades ListaVenta	Ok
09	Usuario	El usuario diseña la colección de entidades ListaProductoCategoria	Ok
10	Usuario	El usuario diseña la colección de entidades ListaProducto	Ok
11	Usuario	El usuario diseña la colección de entidades ListaVentaDetalle	Ok

Tabla 8.2.3. Pasos de la generación del componente de entidad
Fuente: Elaboración Propia

Resultados:

Nro Resultado	Descripción	Observaciones
Resultado Ok	Se diseña sin inconvenientes el modelo especificado en la prueba.	1
Resultado Error	Se diseña con inconvenientes el modelo especificado en la prueba.	0

Tabla 8.2.4. Resultados de la generación del componente de entidad
Fuente: Elaboración Propia

- ✓ **Prueba 02 – Escenario 01 (02-E01):** Prueba de elaboración de entidades sin framework.

Se elaboraron las entidades especificadas en el modelo de prueba partiendo desde cero y desarrollando cada uno de las clases manualmente, estos fueron los resultados:

Paso	Descripción	Tiempo
01	Elaboración de la entidad Cliente	6 min
02	Elaboración de la entidad Venta	5 min
03	Elaboración de la entidad ProductoCategoria	3 min
04	Elaboración de la entidad Producto	5 min
05	Elaboración de la entidad VentaDetalle	5 min
06	Elaboración de la colección de entidades ListaCliente	1 min
07	Elaboración de la colección de entidades ListaVenta	1 min
08	Elaboración de la colección de entidades ListaProductoCategoria	1 min
09	Elaboración de la colección de entidades ListaProducto	1 min
10	Elaboración de la colección de entidades ListaVentaDetalle	1 min
Tiempo Total		29 min

Tabla 8.2.5. Resultados de la generación del componente de entidad - 02-E01
Fuente: Elaboración Propia

- ✓ **Prueba 03 – Escenario 01 (03-E01):** Prueba de elaboración de entidades con framework.

Se elaboraron las entidades especificadas en el modelo de prueba utilizando la herramienta DSL elaborada, estos fueron los resultados:

Paso	Descripción	Tiempo
01	Elaboración de la entidad Cliente	4 min
02	Elaboración de la entidad Venta	3 min
03	Elaboración de la entidad ProductoCategoria	3 min
04	Elaboración de la entidad Producto	3 min
05	Elaboración de la entidad VentaDetalle	3 min
06	Elaboración de la colección de entidades ListaCliente	0.5 min
07	Elaboración de la colección de entidades ListaVenta	0.5 min
08	Elaboración de la colección de entidades ListaProductoCategoria	0.5 min
09	Elaboración de la colección de entidades ListaProducto	0.5 min
10	Elaboración de la colección de entidades ListaVentaDetalle	0.5 min
Tiempo Total		18.5 min

Tabla 8.2.6. Resultados de la generación del componente de entidad - 03-E01
Fuente: Elaboración Propia

Comparación de tiempos: Se han comparado los tiempos de desarrollo de las dos pruebas técnicas:

Nro	Escenario	Descripción	Tiempo (min)
2	02-E01	Prueba de elaboración de entidades sin framework.	29
3	03-E01	Prueba de elaboración de entidades con framework.	18.5

Tabla 8.2.7. Comparación de tiempos de la generación del componente de entidad
Fuente: Elaboración Propia

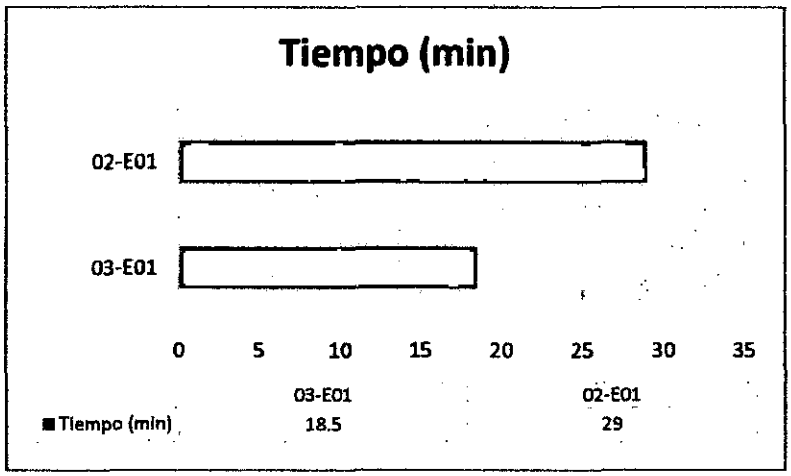


Figura 8.2.2. Comparación de tiempos de la generación del componente de entidad
Fuente: Elaboración Propia

8.3. Pruebas de la generación del componente de acceso a datos

Set de Pruebas

Nro	Tipo de Prueba	Escenario	Descripción
01	Funcional	E01	Prueba de diseño de modelo de entidades - sección de acceso a datos.
02	Técnica	E01	Prueba de elaboración de objetos de base de datos y clases de acceso a datos sin framework.
03	Técnica	E01	Prueba de elaboración de objetos de base de datos y clases de acceso a datos con framework.

Tabla 8.3.1. Set de pruebas de la generación del componente de acceso a datos
Fuente: Elaboración Propia

- ✓ **Prueba 01 – Escenario 01 (01-E01):** Prueba de diseño de modelo de entidades - sección de acceso a datos.

Usuarios:

Usuario	Descripción
Analista Programador	Analista programador encargado de desarrollar el proyecto.

Tabla 8.3.2. Usuarios de la generación del componente de acceso a datos
Fuente: Elaboración Propia

Formato de Prueba: El usuario tendrá que elaborar el siguiente modelo de entidades de dominio del ejemplo de la sección *8.2. Pruebas de la generación del componente de entidad*, en la sección DataAccess de las entidades de dominio y sus propiedades.

Pasos:

Pasos 01-E01			
Nro	Responsable	Descripción	Resultado
01	Usuario	El usuario abre el archivo domainentitymodel.domainentity del proyecto de modelado de la solución elaborada en el punto 8.1 del documento.	Ok
02	Usuario	El usuario diseña la entidad Cliente (Sección Data Access)	Ok
03	Usuario	El usuario diseña la entidad Venta (Sección Data Access)	Ok
04	Usuario	El usuario diseña la entidad ProductoCategoria (Sección Data Access)	Ok
05	Usuario	El usuario diseña la entidad Producto (Sección Data Access)	Ok
06	Usuario	El usuario diseña la entidad VentaDetalle (Sección Data Access)	Ok

Tabla 8.3.3. Pasos de la generación del componente de acceso a datos
Fuente: Elaboración Propia

Resultados:

Tipo Resultado	Descripción	Ejecuciones
Resultado Ok	Se diseña sin inconvenientes el modelo especificado en la prueba.	1
Resultado Error	Se diseña con inconvenientes el modelo especificado en la prueba.	0

Tabla 8.3.4. Resultados de la generación del componente de acceso a datos
Fuente: Elaboración Propia

✓ **Prueba 02 – Escenario 01 (02-E01):** Prueba de elaboración de objetos de base de datos y clases de acceso a datos sin framework.

Se elaboraron las tablas de base de datos, procedimientos almacenados (SP) y clases de acceso a datos para el modelo de prueba partiendo desde cero y desarrollando cada uno de los componentes manualmente, estos fueron los resultados:

Paso	Descripción	Tiempo
01	Elaboración de la Tabla Cliente	6 min
02	Elaboración del SP de Inserción en Tabla Cliente	5 min
03	Elaboración del SP de Actualización en Tabla Cliente	5 min
04	Elaboración del SP de Eliminación en Tabla Cliente	4 min
05	Elaboración del SP de Obtención por Id de Tabla Cliente	4 min
06	Elaboración del SP de Lista de la Tabla Cliente	3 min
07	Elaboración de la clase de acceso a datos ClienteDataAccess	22 min
08	Elaboración de la Tabla Venta	5 min
09	Elaboración del SP de Inserción en Tabla Venta	4 min
10	Elaboración del SP de Actualización en Tabla Venta	4 min
11	Elaboración del SP de Eliminación en Tabla Venta	3 min
12	Elaboración del SP de Obtención por Id de Tabla Venta	4 min
13	Elaboración del SP de Lista de la Tabla Venta	3 min
14	Elaboración de la clase de acceso a datos VentaAccess	19 min
15	Elaboración de la Tabla ProductoCategoria	3 min
16	Elaboración del SP de Inserción en Tabla ProductoCategoria	3 min
17	Elaboración del SP de Actualización en Tabla ProductoCategoria	2 min
18	Elaboración del SP de Eliminación en Tabla ProductoCategoria	2 min
19	Elaboración del SP de Obtención por Id de Tabla ProductoCategoria	3 min
20	Elaboración del SP de Lista de la Tabla ProductoCategoria	3 min
21	Elaboración de la clase de acceso a datos ProductoCategoriaAccess	15 min
22	Elaboración de la Tabla Producto	7 min
23	Elaboración del SP de Inserción en Tabla Producto	6 min
24	Elaboración del SP de Actualización en Tabla Producto	6 min
25	Elaboración del SP de Eliminación en Tabla Producto	4 min
26	Elaboración del SP de Obtención por Id de Tabla Producto	4 min
27	Elaboración del SP de Lista de la Tabla Producto	4 min
28	Elaboración de la clase de acceso a datos ProductoAccess	17 min
29	Elaboración de la Tabla VentaDetalle	6 min
30	Elaboración del SP de Inserción en Tabla VentaDetalle	4 min
31	Elaboración del SP de Actualización en Tabla VentaDetalle	4 min
32	Elaboración del SP de Eliminación en Tabla VentaDetalle	3 min
33	Elaboración del SP de Obtención por Id de Tabla VentaDetalle	3 min
34	Elaboración del SP de Lista de la Tabla VentaDetalle	3 min
35	Elaboración de la clase de acceso a datos VentaDetalle Access	18 min
Tiempo Total		211 min

Tabla 8.3.5. Resultados de la generación del componente de acceso a datos - 02-E01
Fuente: Elaboración Propia

✓ **Prueba 03 – Escenario 01 (03-E01):** Prueba de elaboración de objetos de base de datos y clases de acceso a datos con framework.

Se elaboraron las tablas de base de datos, procedimientos almacenados (SP) y clases de acceso a datos para el modelo de prueba partiendo desde el modelo especificado en el ejemplo, estos fueron los resultados:

Paso	Descripción	Tiempo
01	Elaboración de la Tabla Cliente	6 min
02	Elaboración del SP de Inserción en Tabla Cliente	2 seg
03	Elaboración del SP de Actualización en Tabla Cliente	2 seg
04	Elaboración del SP de Eliminación en Tabla Cliente	2 seg
05	Elaboración del SP de Obtención por Id de Tabla Cliente	2 seg
06	Elaboración del SP de Lista de la Tabla Cliente	2 seg
07	Elaboración de la clase de acceso a datos ClienteDataAccess	2 seg
08	Elaboración de la Tabla Venta	5 min
09	Elaboración del SP de Inserción en Tabla Venta	2 seg
10	Elaboración del SP de Actualización en Tabla Venta	2 seg
11	Elaboración del SP de Eliminación en Tabla Venta	2 seg
12	Elaboración del SP de Obtención por Id de Tabla Venta	2 seg
13	Elaboración del SP de Lista de la Tabla Venta	2 seg
14	Elaboración de la clase de acceso a datos VentaAccess	2 seg
15	Elaboración de la Tabla ProductoCategoria	4 min
16	Elaboración del SP de Inserción en Tabla ProductoCategoria	2 seg
17	Elaboración del SP de Actualización en Tabla ProductoCategoria	2 seg
18	Elaboración del SP de Eliminación en Tabla ProductoCategoria	2 seg
19	Elaboración del SP de Obtención por Id de Tabla ProductoCategoria	2 seg
20	Elaboración del SP de Lista de la Tabla ProductoCategoria	2 seg
21	Elaboración de la clase de acceso a datos ProductoCategoriaAccess	2 seg
22	Elaboración de la Tabla Producto	5 min
23	Elaboración del SP de Inserción en Tabla Producto	2 seg
24	Elaboración del SP de Actualización en Tabla Producto	2 seg
25	Elaboración del SP de Eliminación en Tabla Producto	2 seg
26	Elaboración del SP de Obtención por Id de Tabla Producto	2 seg
27	Elaboración del SP de Lista de la Tabla Producto	2 seg
28	Elaboración de la clase de acceso a datos ProductoAccess	2 seg
29	Elaboración de la Tabla VentaDetalle	6 min
30	Elaboración del SP de Inserción en Tabla VentaDetalle	2 seg
31	Elaboración del SP de Actualización en Tabla VentaDetalle	2 seg
32	Elaboración del SP de Eliminación en Tabla VentaDetalle	2 seg
33	Elaboración del SP de Obtención por Id de Tabla VentaDetalle	2 seg
34	Elaboración del SP de Lista de la Tabla VentaDetalle	2 seg
35	Elaboración de la clase de acceso a datos VentaDetalle Access	2 seg
Tiempo Total		27 min

Tabla 8.3.6. Resultados de la generación del componente de acceso a datos - 03-E01
Fuente: Elaboración Propia

Comparación de tiempos: Se han comparado los tiempos de desarrollo de las dos pruebas técnicas:

Nro	Escenario	Descripción	Tiempo (min)
2	02-E01	Prueba de elaboración de objetos de base de datos y clases de acceso a datos sin framework.	211
3	03-E01	Prueba de elaboración de objetos de base de datos y clases de acceso a datos con framework.	27

Tabla 8.3.7. Comparación de tiempos de la generación del componente de acceso a datos
Fuente: Elaboración Propia

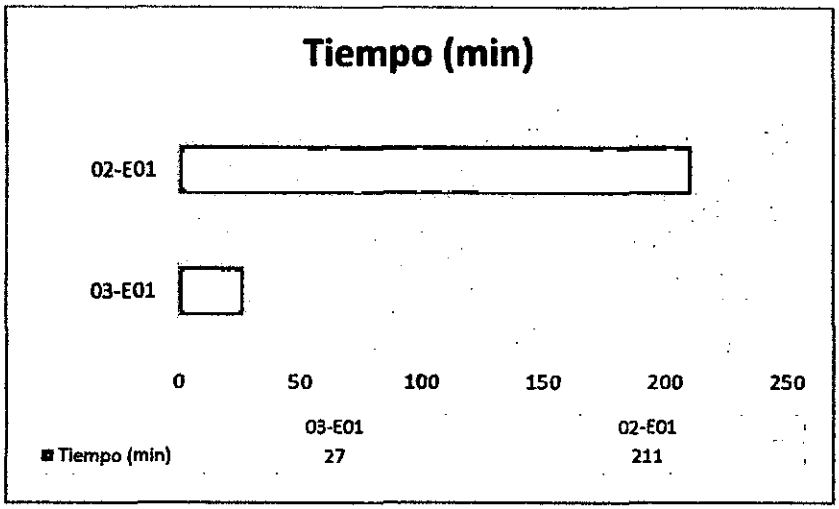


Figura 8.3.1. Comparación de tiempos de la generación del componente de acceso a datos
Fuente: Elaboración Propia

8.4. Pruebas de acoplamiento de la solución

Set de Pruebas

Se ha elaborado el siguiente set de pruebas funcionales para probar el acoplamiento de la solución:

Nro	Tipo de Prueba	Escenario	Descripción
01	Técnica	E01	Prueba de script generado para Cliente.
02	Funcional	E01	Registro de Cliente.
03	Funcional	E01	Actualización de Cliente.
04	Funcional	E01	Eliminación de Cliente.
05	Funcional	E01	Obtención de Cliente por Id.
06	Funcional	E01	Lista de Clientes.

Tabla 8.4.1. Set de pruebas de acoplamiento
Fuente: Elaboración Propia

Para ello se ha agregado un proyecto de Tipo Test C# para probar el componente de acceso a datos generado por ejemplo de las pruebas realizadas en las sección *8.3. Pruebas de la generación del componente de acceso a datos*. El proyecto que se ha agregado a la carpeta de solución Infraestructure > Data es *Enterprise.Module.Data.MainModule.Test*, Se ha agregado un archivo de Test llamado. *ClienteDataAccessUnitTest.cs* para elaborar las pruebas unitarias del componente de acceso a datos *ClienteDataAccess.cs*. La estructura del proyecto se muestra a continuación.

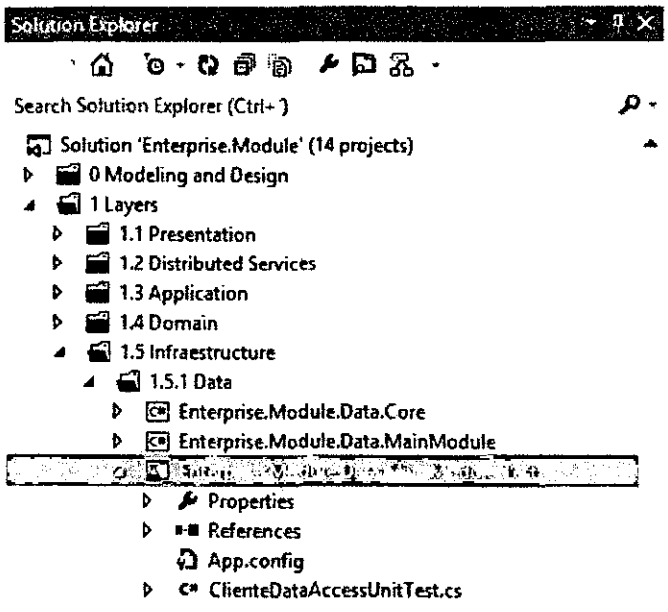


Figura 8.4.1. Proyecto de test para pruebas de acoplamiento
Fuente: Elaboración Propia

✓ **Prueba 01 – Escenario 01 (01-E01):** Prueba de script generado para Cliente.

Datos de Prueba:

Servidor:	NELSSON-PC\MSSQLSERVER2012
Base de Datos:	NLayerDSLTools

Ejecución de Prueba: Se ejecutó el script de creación de objetos en la base de datos.

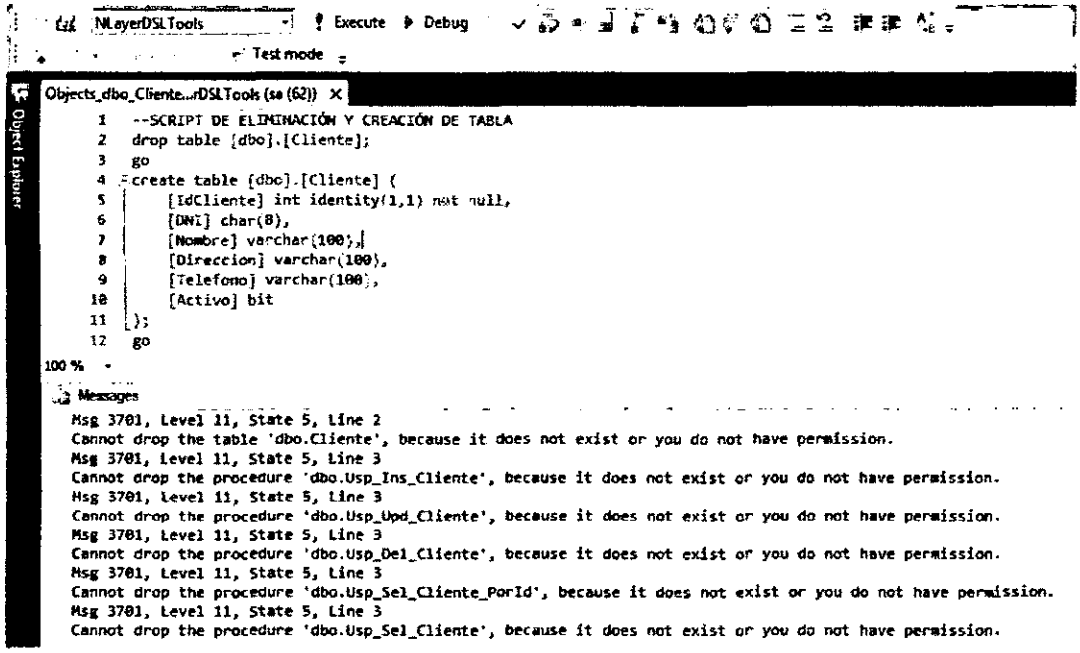


Figura 8.4.2. Ejecución del script de creación de objetos Cliente
Fuente: Elaboración Propia

Resultados: Debido a que es la ejecución inicial del script, y este contiene sentencias DROP y CREATE por cada objeto, se generan errores para las sentencias DROP.
Para el caso de las sentencias CREATE, estas se ejecutan correctamente y se crean la tabla y los procedimientos almacenados para la entidad Cliente:

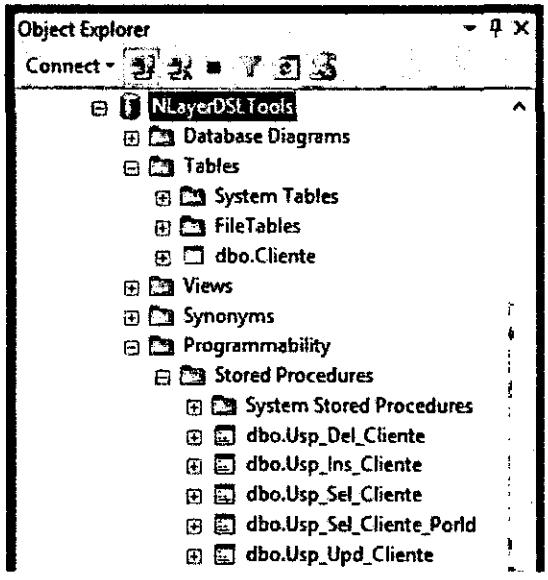


Figura 8.4.3. Generación de objetos de base de datos para Cliente
Fuente: Elaboración Propia

✓ **Prueba 02 – Escenario 01 (02-E01):** Registro de Cliente.

Datos de Prueba:

Servidor: NELSSON-PC\MSSQLSERVER2012
Base de Datos: NLayerDSLTools

Cliente	Dato	Valor
Cliente 01	DNI	46355472
	Nombre	Nelsson José Aguilar Salvador
	Direccion	Calle Arequipa #228 – Frías
	Teléfono	968133858
	Activo	True
Cliente 02	DNI	46355473
	Nombre	Kevin Josué Aguilar Salvador
	Direccion	Calle Arequipa #228 - Frías
	Telefono	968133463
	Activo	True
Cliente 03	DNI	46355474
	Nombre	Lorgio Hildebrando Guarnizo Salvador
	Direccion	Calle Arequipa #226 - Frías
	Telefono	968133582
	Activo	True

Tabla 8.4.2. Datos de prueba para el registro de Cliente
Fuente: Elaboración Propia

Ejecución de Prueba: Se ejecutó la prueba unitaria para el registro de clientes obteniendo el siguiente resultado:

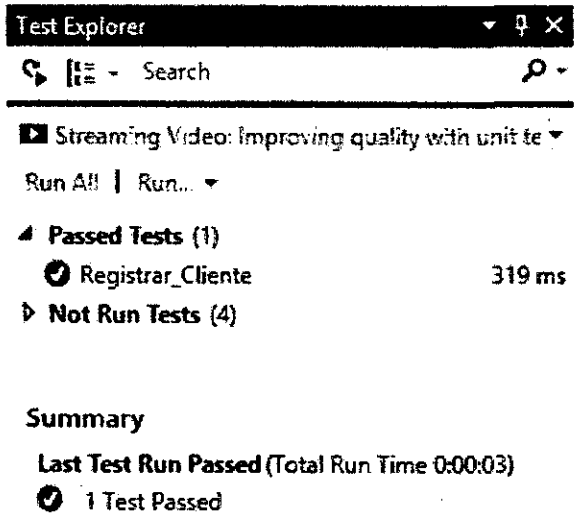
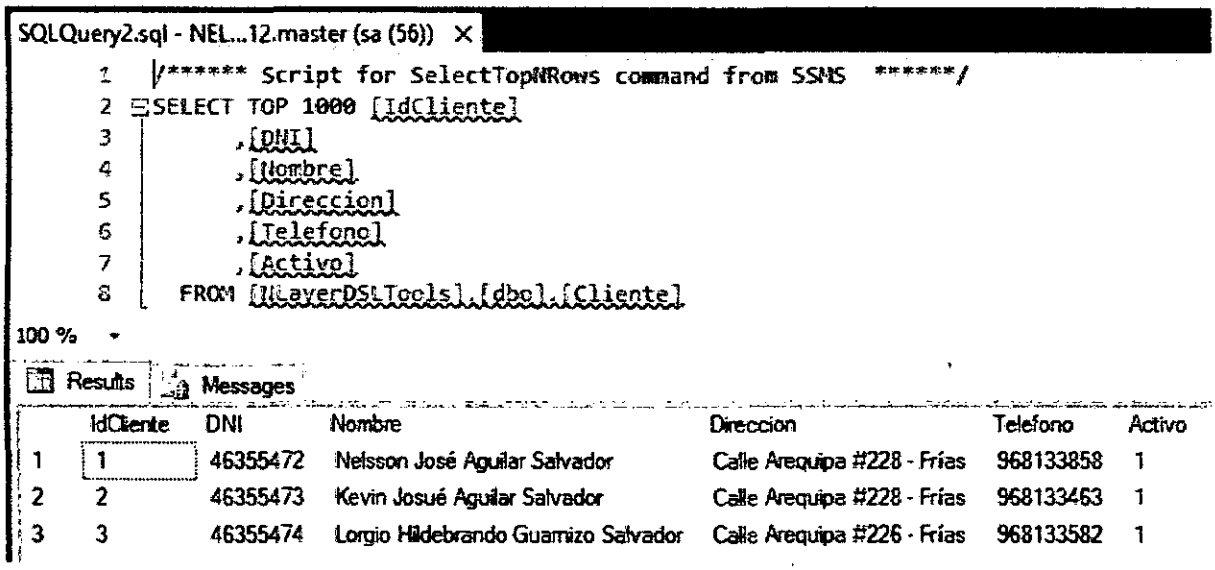


Figura 8.4.4. Ejecución del registro de Cliente - correcto
Fuente: Elaboración Propia

Resultados: Se ejecutó correctamente el registro de clientes en la base de datos:



	IdCliente	DNI	Nombre	Direccion	Telefono	Activo
1	1	46355472	Nelsson José Aguilar Salvador	Calle Arequipa #228 - Frías	968133858	1
2	2	46355473	Kevin Josué Aguilar Salvador	Calle Arequipa #228 - Frías	968133463	1
3	3	46355474	Lorgio Hildebrando Guamizo Salvador	Calle Arequipa #226 - Frías	968133582	1

Figura 8.4.5. Ejecución del registro de Cliente en base de datos - correcto
Fuente: Elaboración Propia

✓ **Prueba 03 – Escenario 01 (03-E01):** Actualización de Cliente.

Datos de Prueba:

Servidor: NELSSON-PC\MSSQLSERVER2012
Base de Datos: NLayerDSLTools

Cliente	Dato	Valor
Cliente 01	IdCliente	1
	DNI	46355472
	Nombre	Nelsson José Aguilar Salvador
	Dirección	Av. Gerardo Unger #1637 - Lima
	Telefono	968133858
	Activo	True

Tabla 8.4.3. Datos de prueba para la actualización de Cliente
Fuente: Elaboración Propia

Ejecución de Prueba: Se ejecutó la prueba unitaria para la actualización de clientes obteniendo el siguiente resultado:

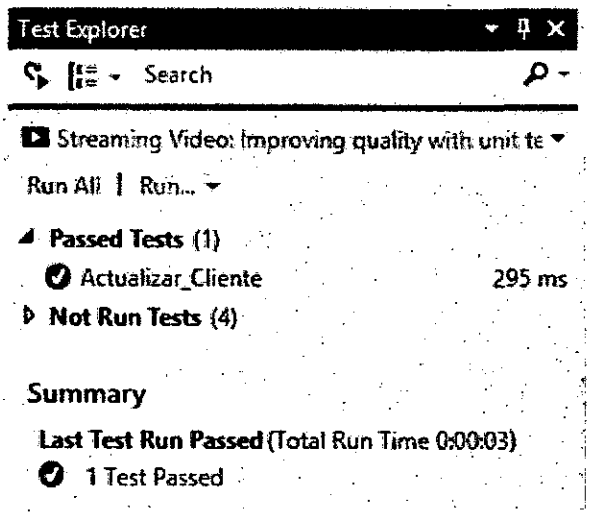


Figura 8.4.6. Ejecución de la actualización de Cliente - correcta
Fuente: Elaboración Propia

Resultados: Se ejecutó correctamente la actualización de clientes en la base de datos:

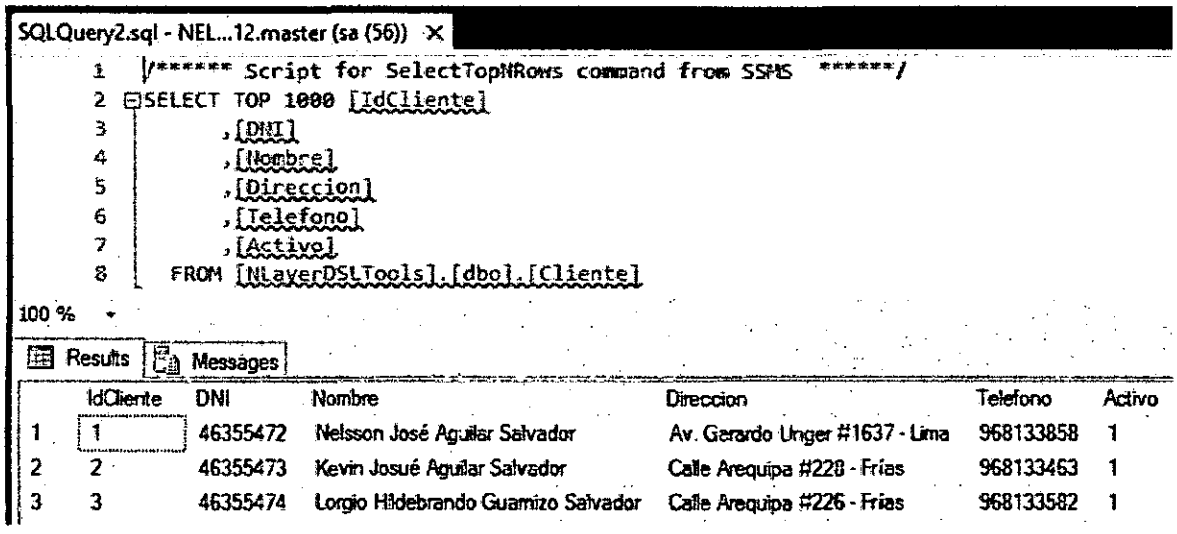


Figura 8.4.7. Ejecución de la actualización de Cliente en base de datos - correcta

Fuente: Elaboración Propia

✓ **Prueba 04 – Escenario 01 (04-E01):** Eliminación de Cliente.

Datos de Prueba:

Servidor: NELSSON-PC\MSSQLSERVER2012
Base de Datos: NLayerDSLTools

Cliente	Dato	Valor
Cliente 01	IdCliente	1

Tabla 8.4.4. Datos de prueba para la eliminación de Cliente
Fuente: Elaboración Propia

Ejecución de Prueba: Se ejecutó la prueba unitaria para la eliminación de cliente obteniendo el siguiente resultado:

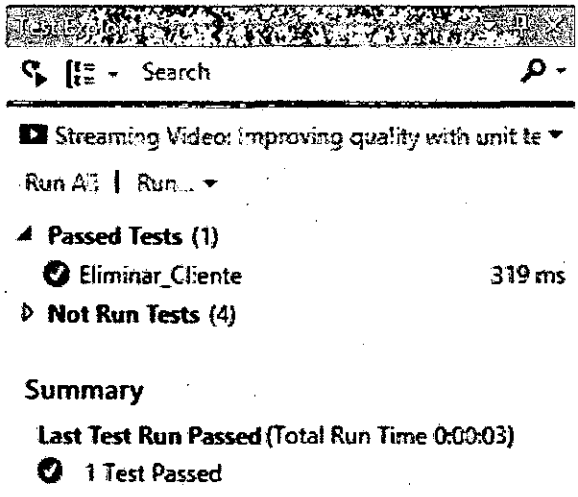


Figura 8.4.8. Ejecución de la eliminación de Cliente - correcta
Fuente: Elaboración Propia

Resultados

Se ejecutó correctamente la eliminación de clientes en la base de datos:

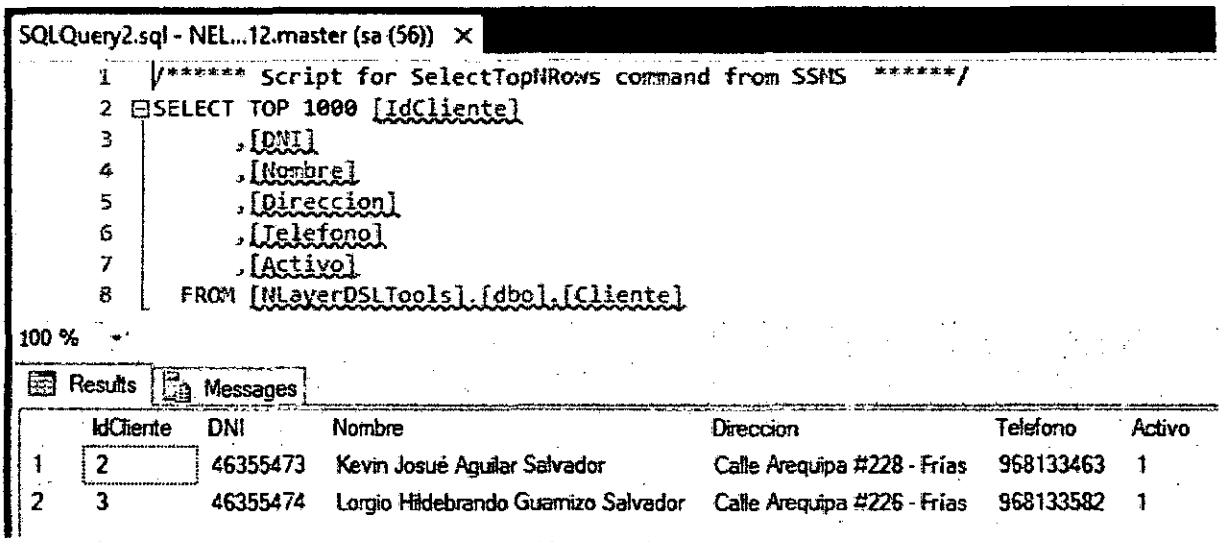


Figura 8.4.9. Ejecución de la eliminación de Cliente en base de datos - correcta
Fuente: Elaboración Propia

✓ **Prueba 05 – Escenario 01 (05-E01):** Obtención de Cliente por Id.

Datos de Prueba:

Servidor: NELSSON-PC\MSSQLSERVER2012
Base de Datos: NLayerDSLTools

Cliente	Dato	Valor
Cliente 01	IdCliente	2

Tabla 8.4.5. Datos de prueba para la obtención de Cliente por Id
Fuente: Elaboración Propia

Ejecución de Prueba: Se ejecutó la prueba unitaria para la obtención de cliente por id obteniendo el siguiente resultado:

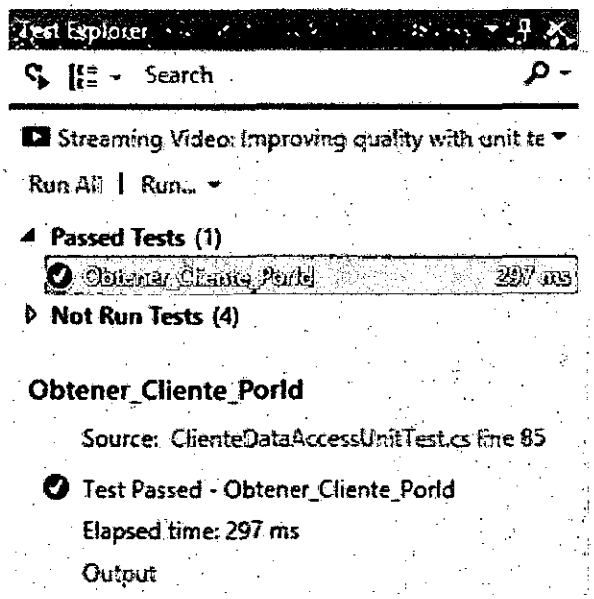


Figura 8.4.10. Ejecución de la obtención de Cliente por id - correcta
Fuente: Elaboración Propia

Resultados

Los resultados de salida para la obtención de cliente por id desde la base de datos del escenario son los siguientes:

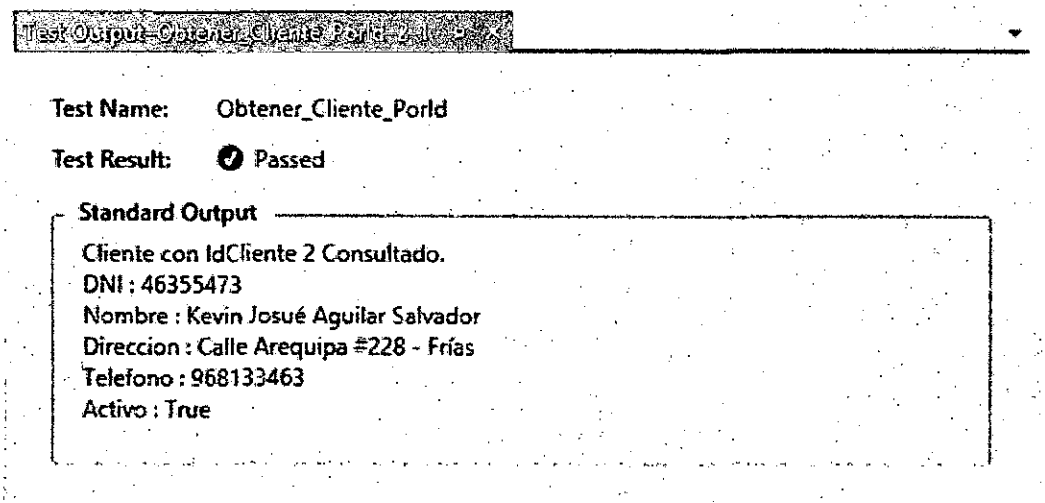


Figura 8.4.11. Ejecución de la obtención de Cliente desde la base de datos - correcta
Fuente: Elaboración Propia

✓ **Prueba 06 – Escenario 01 (06-E01):** Lista de Clientes.

Datos de Prueba:

Servidor: NELSSON-PC\MSSQLSERVER2012
Base de Datos: NLayerDSLTools

Cliente	Dato	Valor
Cliente 01	Ninguno	No se necesitan filtros.

Tabla 8.4.6. Datos de prueba para la eliminación de Cliente
Fuente: Elaboración Propia

Ejecución de Prueba: Se ejecutó la prueba unitaria para el listado de clientes obteniendo el siguiente resultado:

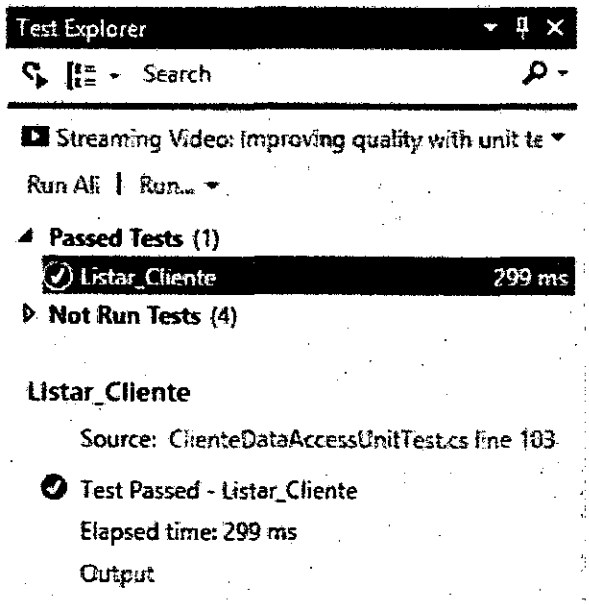


Figura 8.4.12. Ejecución del listado de Clientes – correcto
Fuente: Elaboración Propia

Resultados

Los resultados de salida para el listado de clientes desde la base de datos del escenario los siguientes:

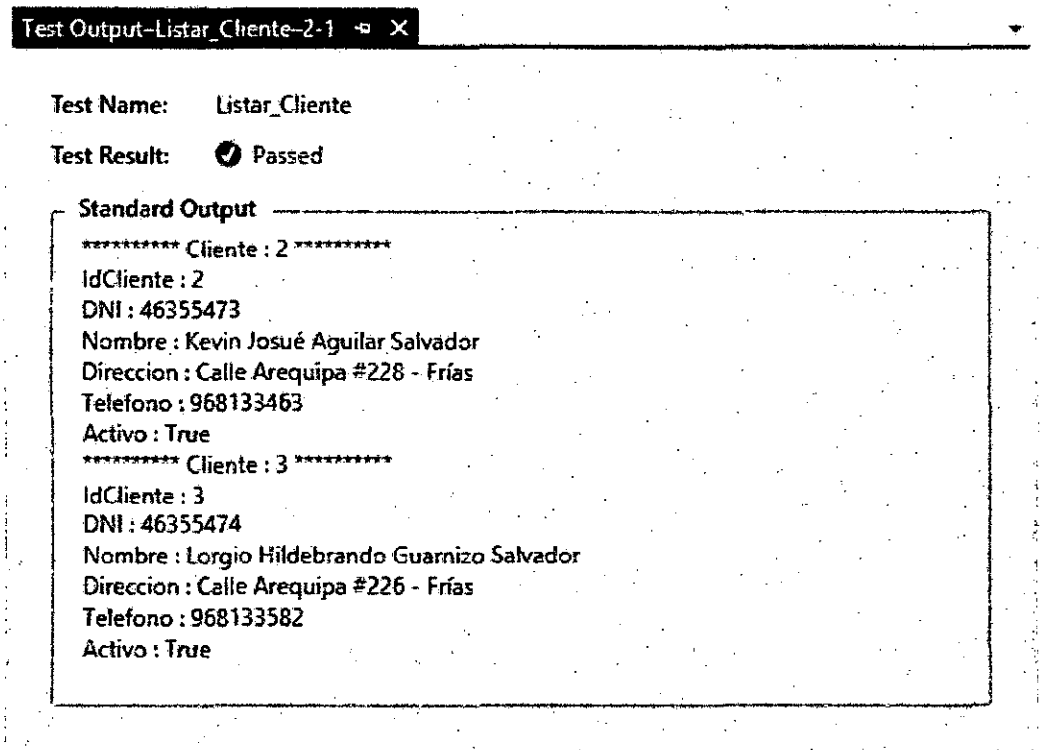


Figura 8.4.13. Ejecución del listado de Clientes desde la base de datos - correcto
Fuente: Elaboración Propia

8.5. Contrastación de Hipótesis

Hipótesis de la Investigación e Hipótesis Nula

Para poder contrastar la hipótesis de investigación y la hipótesis nula se han obtenido los siguientes resultados de las pruebas realizadas al framework.

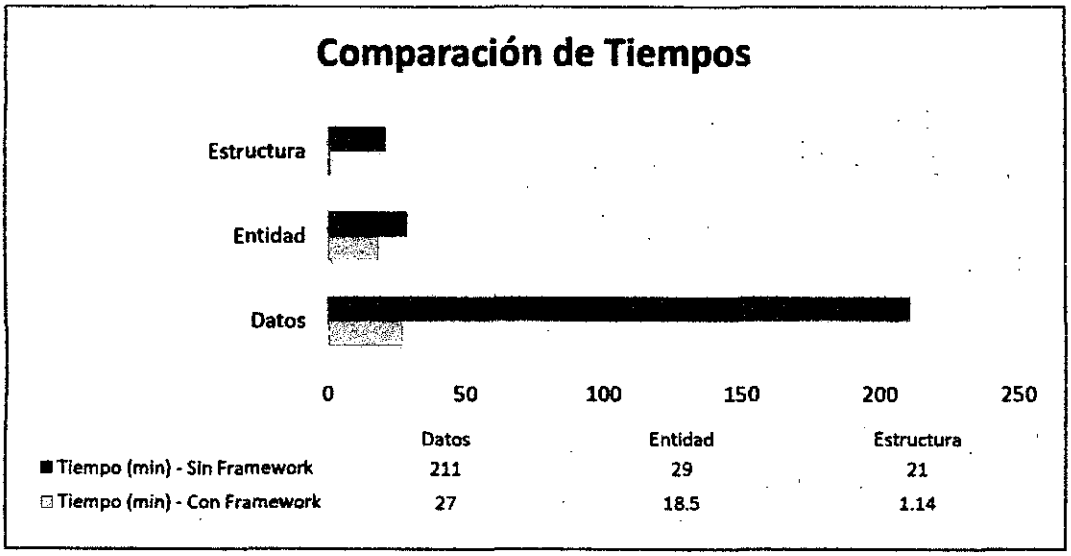


Figura 8.5.1. Comparación de tiempos de desarrollo
Fuente: Elaboración Propia

Hipótesis de la Investigación:
Hi: El uso de las mejoras en el framework N-Capas orientado al dominio permitirá reducir los tiempos de desarrollo del aplicativo.

Hipótesis Nula:
Ho: El uso de las mejoras en el framework N-Capas orientado al dominio no permitirá reducir los tiempos de desarrollo del aplicativo

Para demostrar la reducción de tiempos medidos se consideran el desarrollo del componente usando las mejoras en el framework y sin usar mejoras en el framework:

T_{SF}: Tiempo de desarrollo del *componente* sin mejoras en el framework
T_{CF}: Tiempo de desarrollo de *componente* con mejoras en el framework

Y el porcentaje de reducción de tiempo se calcula de la siguiente forma:

$$P_{RT} = 100 - ((T_{CF} * 100) / T_{SF})$$

Si $P_{RT} > 0$ % se habrá reducido el tiempo de desarrollo.

Para el componente *plantilla del framework*:

T_{SF}: 21 min **T_{CF}:** 1.14 min

P_{RT} = 100 - ((1.14 * 100) / 21) = 94.57 % > 0 %

Para el componente *entidad*:

T_{SF}: 29 min **T_{CF}:** 18.5 min

P_{RT} = 100 - ((18.5 * 100) / 29) = 36.21 % > 0 %

Para el componente de *base de datos*:

T_{SF}: 221 min **T_{CF}:** 27 min

P_{RT} = 100 - ((27 * 100) / 221) = 87.78 % > 0 %

De acuerdo las conclusiones sobre tiempos se puede observar que han disminuido considerablemente los tiempos de desarrollo tanto de la arquitectura del proyecto como de los componentes de entidad y objetos de base de datos por lo que queda demostrada la hipótesis de investigación.

Hipótesis Alternativa:

Para demostrar la hipótesis alternativa se tomarán los datos obtenidos en las pruebas de *Pruebas de la generación del componente de acceso a datos (Sesión 8.3 del documento)* tomando en cuenta las funcionalidades implementadas en la prueba, las funcionalidades y los tiempos tomados haciendo y no haciendo uso del framework se muestran en el cuadro siguiente:

Tiempo SF: Tiempo de desarrollo sin mejoras en el framework
Tiempo CF: Tiempo de desarrollo con mejoras en el framework

Paso	Descripción	Tiempo SF	Tiempo CF
01	Elaboración de la Tabla Cliente	6 min	6 min
02	Elaboración del SP de Inserción en Tabla Cliente	5 min	2 seg
03	Elaboración del SP de Actualización en Tabla Cliente	5 min	2 seg
04	Elaboración del SP de Eliminación en Tabla Cliente	4 min	2 seg
05	Elaboración del SP de Obtención por Id de Tabla Cliente	4 min	2 seg
06	Elaboración del SP de Lista de la Tabla Cliente	3 min	2 seg
07	Elaboración de la clase de acceso a datos ClienteDataAccess	22 min	2 seg
08	Elaboración de la Tabla Venta	5 min	5 min
09	Elaboración del SP de Inserción en Tabla Venta	4 min	2 seg
10	Elaboración del SP de Actualización en Tabla Venta	4 min	2 seg
11	Elaboración del SP de Eliminación en Tabla Venta	3 min	2 seg
12	Elaboración del SP de Obtención por Id de Tabla Venta	4 min	2 seg
13	Elaboración del SP de Lista de la Tabla Venta	3 min	2 seg
14	Elaboración de la clase de acceso a datos VentaAccess	19 min	2 seg
15	Elaboración de la Tabla ProductoCategoria	3 min	4 min
16	Elaboración del SP de Inserción en Tabla ProductoCategoria	3 min	2 seg
17	Elaboración del SP de Actualización en Tabla ProductoCategoria	2 min	2 seg
18	Elaboración del SP de Eliminación en Tabla ProductoCategoria	2 min	2 seg
19	Elaboración del SP de Obtención por Id de Tabla ProductoCategoria	3 min	2 seg
20	Elaboración del SP de Lista de la Tabla ProductoCategoria	3 min	2 seg
21	Elaboración de la clase de acceso a datos ProductoCategoriaAccess	15 min	2 seg
22	Elaboración de la Tabla Producto	7 min	5 min

23	Elaboración del SP de Inserción en Tabla Producto	6 min	2 seg
24	Elaboración del SP de Actualización en Tabla Producto	6 min	2 seg
25	Elaboración del SP de Eliminación en Tabla Producto	4 min	2 seg
26	Elaboración del SP de Obtención por Id de Tabla Producto	4 min	2 seg
27	Elaboración del SP de Lista de la Tabla Producto	4 min	2 seg
28	Elaboración de la clase de acceso a datos ProductoAccess	17 min	2 seg
29	Elaboración de la Tabla VentaDetalle	6 min	6 min
30	Elaboración del SP de Inserción en Tabla VentaDetalle	4 min	2 seg
31	Elaboración del SP de Actualización en Tabla VentaDetalle	4 min	2 seg
32	Elaboración del SP de Eliminación en Tabla VentaDetalle	3 min	2 seg
33	Elaboración del SP de Obtención por Id de Tabla VentaDetalle	3 min	2 seg
34	Elaboración del SP de Lista de la Tabla VentaDetalle	3 min	2 seg
35	Elaboración de la clase de acceso a datos VentaDetalle Access	18 min	2 seg
Tiempo Total		211 min	27 min

Tabla 8.5.1. Funcionalidades desarrolladas y tiempos de desarrollo
Fuente: Elaboración Propia

Teniendo en cuenta que por cada entidad de dominio se desarrollan 5 funcionalidades:

- ✓ Registrar entidad de dominio.
- ✓ Actualizar entidad de dominio.
- ✓ Eliminar entidad de dominio.
- ✓ Obtener entidad de dominio por Id.
- ✓ Listar entidad de dominio.

Y resumiendo la tabla para tener una visión de las funcionalidades realizadas para cada una de las entidades del dominio.

Paso	Descripción	TiempoSF	TiempoCF
01	Funcionalidades para Cliente (5 funcionalidades)	49 min	6.2 min
08	Funcionalidades para Venta (5 funcionalidades)	42 min	5.2 min
15	Funcionalidades para ProductoCategoria (5 funcionalidades)	31 min	4.2 min
16	Funcionalidades para Producto (5 funcionalidades)	48 min	5.2 min
17	Funcionalidades para VentaDetalle (5 funcionalidades)	41 min	6.2 min
Tiempo Total (minutos)		211 min	27 min
Tiempo Total (horas)		3.52 hor	0.45 hor

Tabla 8.5.2. Resumen de funcionalidades desarrolladas y tiempos de desarrollo
Fuente: Elaboración Propia

Hipótesis Alternativas:
H1: El uso de las mejoras en el framework N-Capas orientado al dominio permitirá aumentar la productividad de los desarrolladores.

Considerando que la productividad se mide de la siguiente manera:

Productividad *= (Numero de Funcionalidades Desarrolladas / Tiempo de Desarrollo Tomado)*

Para el desarrollo sin el uso de las mejoras en el framework:

Productividad *= (25 funcionalidades/ 3.52 horas) = 7.10 funcionalidades/horas*

Para el desarrollo con el uso de las mejoras en el framework:

Productividad *= (25 funcionalidades/ 0.45 horas) = 55.55 funcionalidades/horas*

Por lo que queda demostrado que la productividad aumenta haciendo uso de las mejoras en el framework y probada la hipótesis alternativa.

8.6. Viabilidad del uso del Framework

De acuerdo a los resultados obtenidos en la presente investigación se puede decir que la aplicación de las mejoras aplicadas a la arquitectura de N-Capas orientado al dominio utilizando tecnologías DSL puede contribuir significativamente en los tiempos de desarrollo tanto al inicio del proyecto como durante el desarrollo influyendo de esta manera en las fechas de entrega del producto a los clientes y usuarios finales.

El uso de estas mejoras es viable para este tipo de arquitectura y también para arquitecturas basadas en tecnología Microsoft ya que la generación de artefactos se acopla fácilmente con otras estructuras de solución. Con respecto a la tecnología usada para implementar estas mejoras puede decirse que debido a la versatilidad que posee es viable implementar modelos no solo para las capas en las que se aplicado en el presente proyecto sino para las demás capas de la arquitectura.

CONCLUSIONES

De la investigación se puede concluir lo siguiente:

- ✓ Se logró realizar la implementación de una solución de extensibilidad de Visual Studio para lenguajes específicos de dominio (DSL) con la finalidad de modelar el diagrama de entidades para la capa de dominio de la arquitectura de N-Capas propuesto por Microsoft.

- ✓ Se logró demostrar que utilizando la solución basada en DSL y Plantillas de Visual Studio se puede reducir el tiempo de elaboración de la estructura de solución de la arquitectura N-Capas en un 94.57%, el desarrollo de las entidades en un 36.21% y el desarrollo de los componentes de datos y acceso a datos en un 87.78% de manera que, basándose en los resultados obtenidos, se puede aumentar la productividad en de 7.1 a 55.55 funcionalidades por hora, beneficiando tanto al arquitecto de software y al desarrollador encargado de elaborar la arquitectura al inicio del proyecto y al desarrollador durante el desarrollo del proyecto.

- ✓ Se pudo demostrar también que se puede automatizar la generación de los componentes de la capa de dominio, la capa de infraestructura de persistencia de datos de la arquitectura N-Capas y los objetos de base de datos, de manera que se pueda reducir el tiempo de los desarrolladores durante el proyecto.

- ✓ Se lograron realizar de manera satisfactoria las pruebas funcionales y pruebas técnicas de manera que se puede liberar una versión release para ser instalada en equipos en los que se desee utilizar la solución elaborada en el presente proyecto.

- ✓ Se demostró la utilidad de la tecnología de extensibilidad provista por Microsoft para generar artefactos, en este caso para las capas de dominio e infraestructura de datos, pero también podría ser de utilidad para generar artefactos para las demás capas como la capa de servicios distribuidos, capa de aplicación e incluso la capa de presentación.

RECOMENDACIONES

Del estudio elaborado se pueden dar las siguientes recomendaciones:

- ✓ Se puede utilizar el presente trabajo de investigación como base para proyectos futuros en los que se piense utilizar la tecnología de lenguaje especifico de dominio (DSL), no solo para proyectos basados en tecnología Microsoft sino también en otras tecnologías que permitan lenguajes de alto nivel para generación de código.

- ✓ Se puede tomar como base el presente proyecto de investigación para futuros proyectos en los que se tengan que utilizar Plantillas de Visual Studio para la generación de la plantilla de un proyecto no solo de la arquitectura de N-Capas orientada al dominio sino de las diversas arquitecturas empleadas para desarrollar software usando esta plataforma.

- ✓ Si bien las mejoras en el framework fueron efectivas para generar artefactos para la arquitectura de N-Capas orientada al dominio, se puede generar código para cualquier otro tipo de estructura de proyecto, ya que se puede adecuar el diagrama de entidades de dominio especificando los nombres de los proyectos en los que se va a generar los artefactos.

- ✓ La arquitectura del proyecto está destinada a ser utilizada en empresas que sean partners de Microsoft o utilicen tecnología de la corporación, se recomienda leer la guía de Arquitectura de N-Capas Orientada al dominio sobre la cual se basó esta investigación para tener una visión más amplia de la arquitectura y que esta solución sea utilizada de la mejor manera.

- ✓ Los componentes necesarios para poder implementar la solución son los siguientes: Visual Studio 2012, Visual Studio 2012 SDK (Software Development Kit), y el motor de base de datos SQL Server 2012, que es el motor que se usó para probar la generación de código de base de datos, y debido a que el código de base de datos generado es nativo (es decir que no tiene ninguna característica especifica de una versión de SQL Server) se puede usar el código generado para versiones de SQL Server 2005, 2008, 2008 R2 y 2014.

BIBLIOGRAFÍA

Bass, L. Clement, P., et al. (2003). *Software Architecture In Practice, 2nd Edition.* Boston: Addison-Wesley.

Baumer, D., Grychzan, G., et al. (1997). *Framework development for large systems.* ACM 40(10), 52-9.

Biggerstaff, T. (1998). *Annals Of Software Engineering.* 5, 169-226.

De la Torre Llorente, C. & Ramos Barroso, M. (Eds.). (2010). *Guía de Arquitectura N-Capas Orientada al Dominio con .NET 4.0.* [Versión de Microsoft Ibérica].

Dobbe, J. (2006). *A Domain-Specific Language for Computer Games.* (Tesis inédita de maestría) Delft University of Technology, Delft, The Netherlands.

Fayad, M.E. and Schmidt. (1997). *Object-Oriented Application Frameworks.* ACM 40(10), 32-8.

Fowler, M. (2003). *Patterns of Application Architecture.* Boston: Addison-Wesley.

Hofmeister, C. Nord, R., et al. (2000). *Applied Software Architecture.* Boston: Addison-Wesley.

Jocobson, I. (Eds.) (2000). *El Proceso unificado de desarrollo de software.* España: Pearson Education.

Mak, S. (2007). *A Domain-Specific Language for Computer Games.* (Tesis inédita de maestría) Universiteit Utrecht, Utrecht, The Netherlands.

Pressman, R. (Eds.) (2010). *Ingeniería del software - Un enfoque práctico.* México: Mc Graw Hill.

Rosén, K. (2013). *A Domain-Specific Language for Cross-Platform Smartphone Application Development.* (Tesis inédita de maestría) Lund University, Lund, Scania, Sweden.

Rumbaugh, J. (Eds.) (2000). *El Lenguaje unificado de modelado – Manual de Referencia.* España: Pearson Education.

Sommerville, I. (Eds.) (2005). *Ingeniería del software.* España: Pearson Education.

Wirfs- Brock, R.J. and Johnson. (1990). *Surveying current research in object oriented design.* ACM 33(9), 104-24.

The Repository Pattern. En Microsoft Developer Network MSDN. Portal Recuperado de https://msdn.microsoft.com/en-us/library/ff649690.aspx

Acerca de los lenguajes específicos de dominio. En Microsoft Developer Network MSDN Portal. Recuperado de https://msdn.microsoft.com/es-es/library/bb126278.aspx

Información general y conceptual sobre .NET Framework. En Microsoft Developer Network MSDN. Portal Recuperado de https://msdn.microsoft.com/es-pe/library/vstudio/zw4w595w(v=vs.100).aspx

Microsoft SQL Server. En Microsoft Developer Network MSDN. Portal Recuperado de https://msdn.microsoft.com/es-pe/library/bb545450.aspx

ANEXOS

ANEXO A

Manual de usuario

1. INSTALACIÓN DEL COMPONENTE

Requerimientos de Software: Se requieren los siguientes componentes para instalar el complemento de extensibilidad de Visual Studio.

- ✓ Microsoft Visual Studio Ultimate 2012 (en-us)
  https://www.microsoft.com/en-us/download/details.aspx?id=30678
- ✓ Microsoft Visual Studio 2012 SDK (en-us)
  https://www.microsoft.com/en-us/download/details.aspx?id=30668
- ✓ Microsoft® SQL Server® 2012 Express (en-us)
  https://www.microsoft.com/en-us/download/details.aspx?id=29062
  Opcional si se quiere ejecutar los scripts generados.

La instalación de la herramienta N-Layer DSL Tools se realiza mediante el instalador de extensiones de Visual Studio 2012, que es un instalador exclusivo para el IDE, no se trata de un archivo .exe o .msi, sino de un archivo de paquete .vsix que es un archivo que contiene la información y ensamblados necesarios para instalar la extensión DSL.

Para instalar la extensión siga los siguientes pasos.

Paso 01: Ubique el archivo de instalación en la carpeta correspondiente, el archivo es *NikkoFramework.NLayerDSLTools.DslPackage.vsix.*

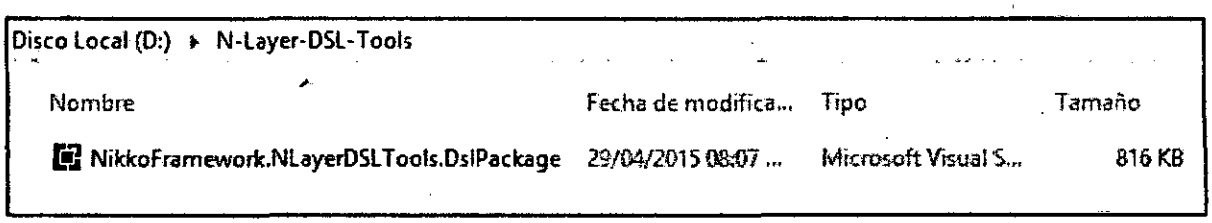


Figura 12.1.1. Archivo de instalación de la extensión N-Layer-DSL-Tools
Fuente: Elaboración Propia

Paso 02: Ejecute el archivo *NikkoFramework.NLayerDSLTools.DslPackage.vsix*, saldrá la siguiente ventana.

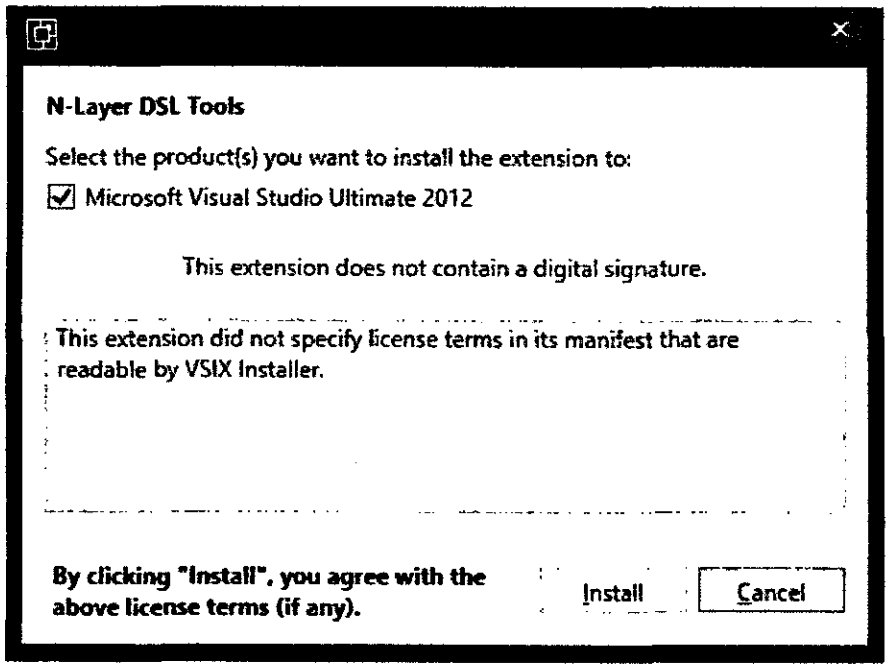


Figura 12.1.2. Ventana inicial de instalación de la extensión N-Layer-DSL-Tools
Fuente: Elaboración Propia

Paso 03: Hacer click en *Install*, le saldrá la siguiente ventana, hacer click en *Close*.

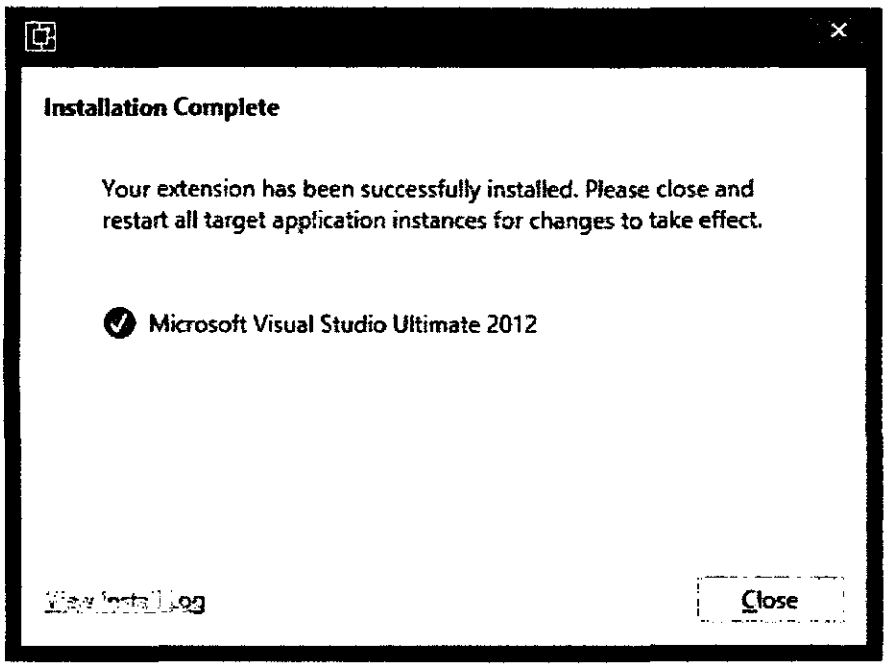


Figura 12.1.3. Ventana final de instalación de la extensión N-Layer-DSL-Tools
Fuente: Elaboración Propia

Paso 04: Para verificar que se ha instalado correctamente el componente abra el Visual Studio 2012, Click en el menú Tools → Extensions and Updates. Se debe listar

el complemento *N-Layer DSL Tools* en la pestaña *All*, como se muestra en la siguiente figura.

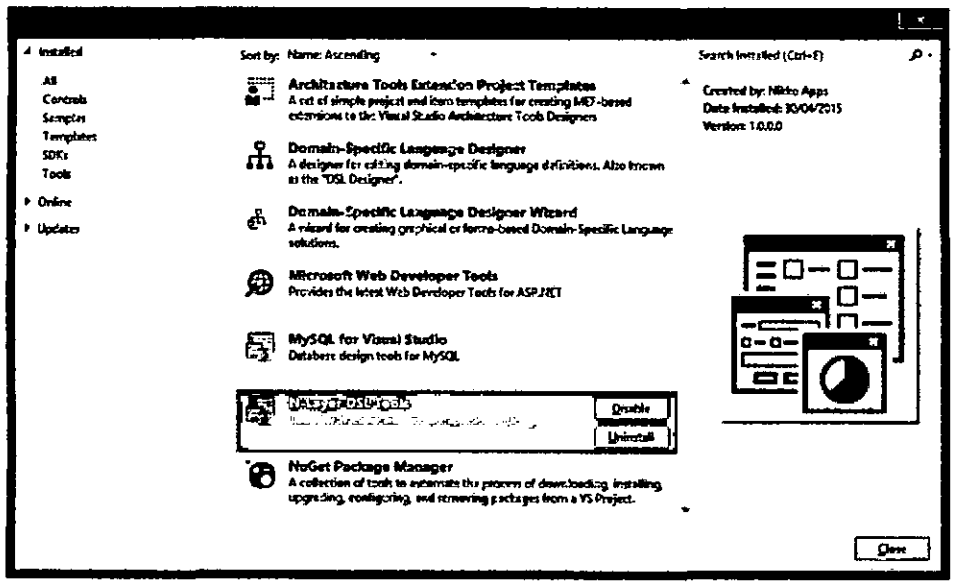


Figura 12.1.4. Verificación de instalación de la extensión N-Layer-DSL-Tools
Fuente: Elaboración Propia

2. CREACIÓN DE UNA SOLUCIÓN DE N-CAPAS

Para crear una solución de N-Capas Orientado al dominio siga los siguientes pasos:

Paso 01: Abrir el IDE Visual Studio 2012 y seleccionar el Menú File → New → Project, se mostrará la siguiente ventana:

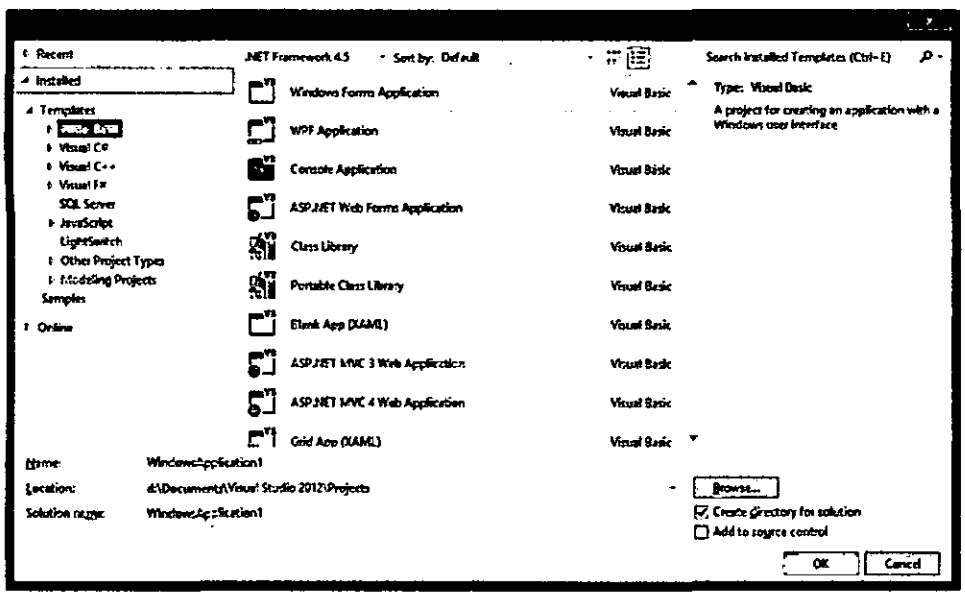


Figura 12.1.5. Creación de un nuevo proyecto
Fuente: Elaboración Propia

Paso 02: Situarse sobre la pestaña Templates → Visual C#, y buscar la plantilla de proyecto denominada *N-Layered Domain-Oriented Architecture Project.*

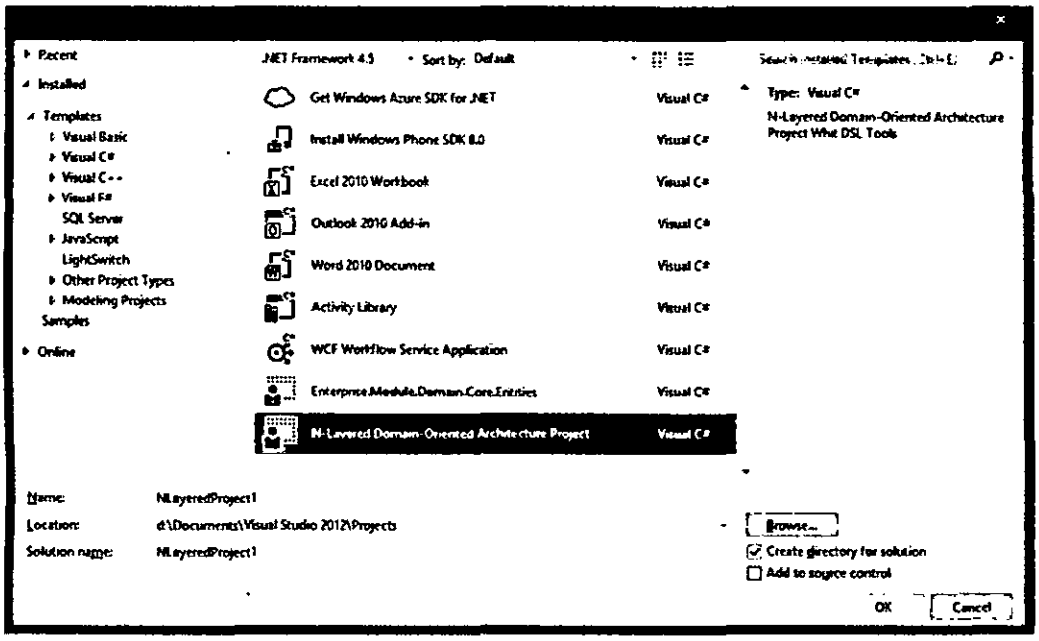


Figura 12.1.6. Selección del tipo de proyecto de N-Capas
Fuente: Elaboración Propia

Paso 03: Establecer un nombre para el proyecto, un nombre para la solución y la ruta de la ubicación y haga click en Ok, para el ejemplo se ha establecido "Enterprise.Module" y dejado la ruta de la ubicación por defecto.

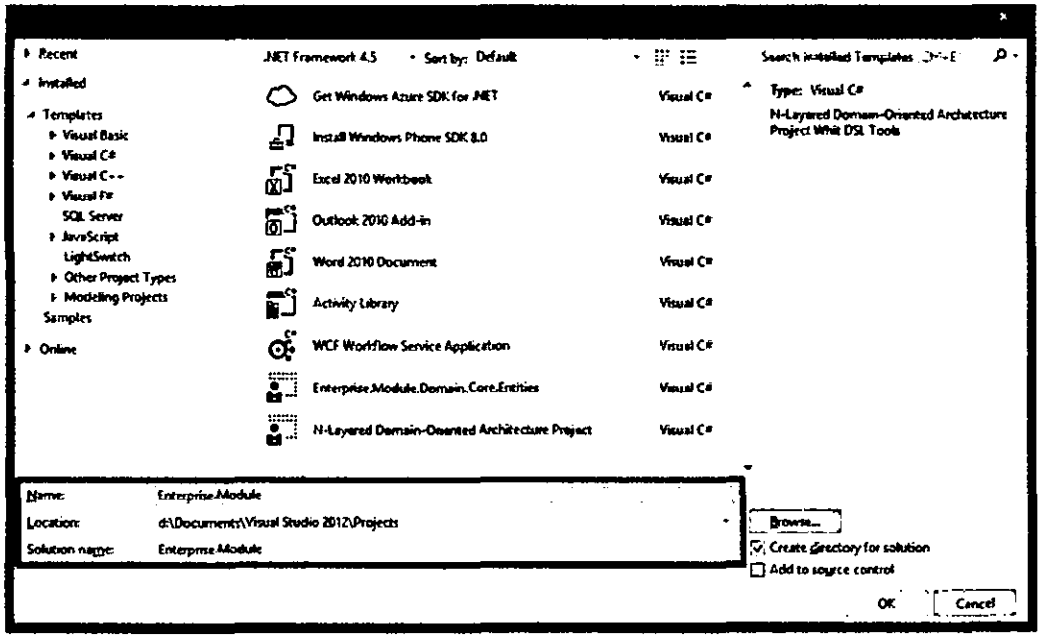


Figura 12.1.7. Establecer el nombre del proyecto de N-Capas
Fuente: Elaboración Propia

Paso 04: Se generará un proyecto con la estructura N-Layer, para verificarlo ubíquese en la ventana "Solution Explorer", tal y como se muestra en el grafico siguiente.

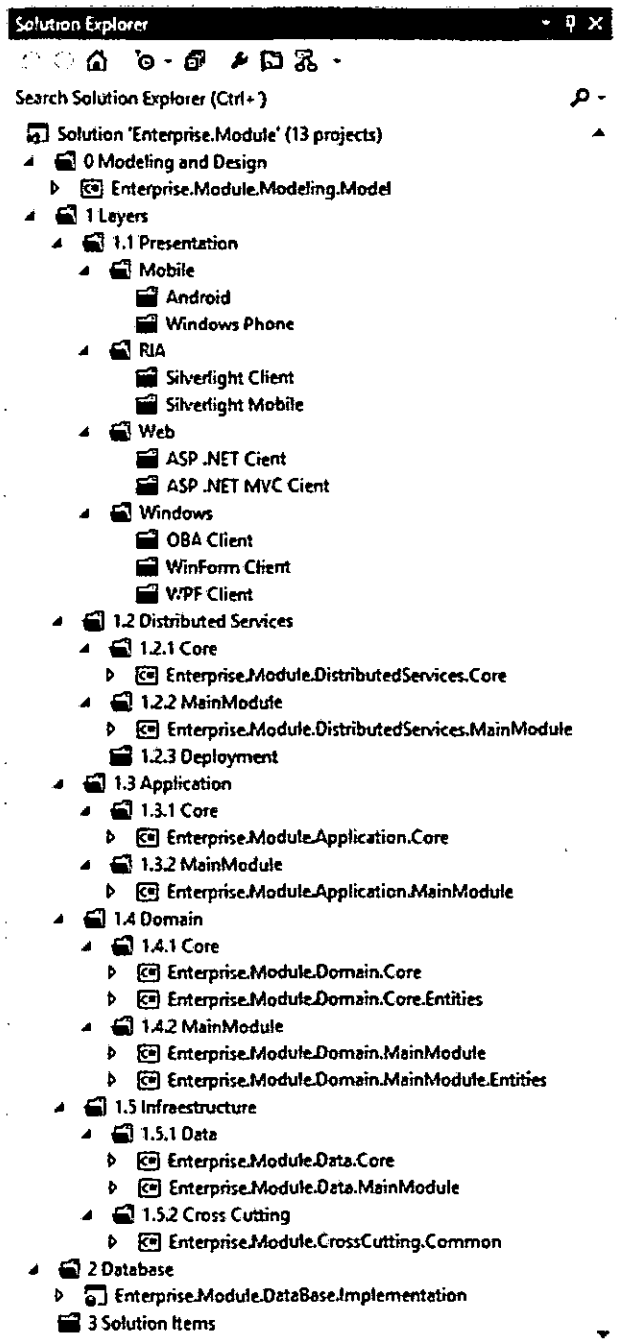


Figura 12.1.8. Proyecto de N-Capas generado
Fuente: Elaboración Propia

3. MODELAR LA CAPA DE ENTIDADES DE DOMINIO

Ubicar el archivo *domainentitymodel.domainentity*, este archivo se encuentra en el proyecto de modelado de la solución, el proyecto tiene el formato de nombre *[NombreSolucion].Modeling.Model* y se encuentra dentro de la carpeta *0 Modeling and Model* de la solución como se muestra en la siguiente figura.

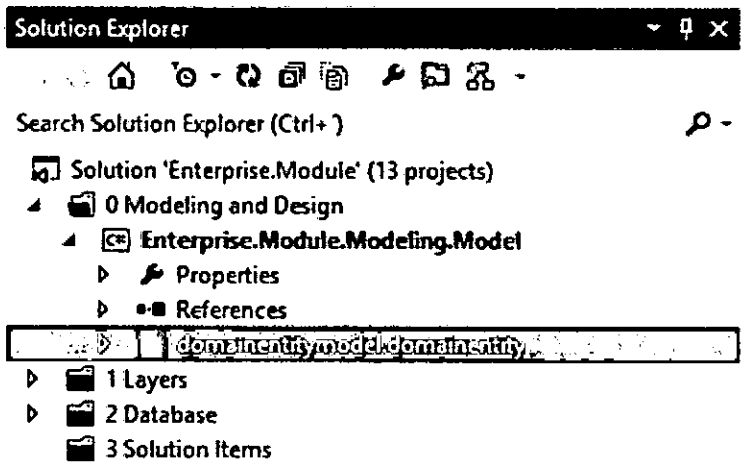


Figura 12.1.9. Archivo de modelado de la capa de entidad de dominio
Fuente: Elaboración Propia

Paso 02: Abrir el archivo, se mostrará el área para el diagrama y se mostrarán las herramientas de diagrama en el cuadro de herramientas (Toolbox).

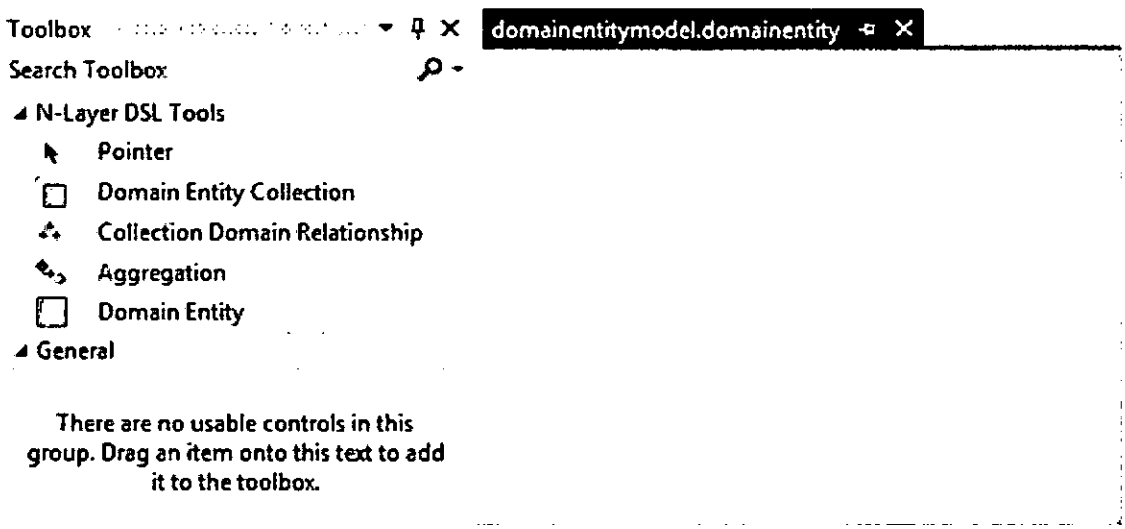


Figura 12.1.10. Diagrama de modelado de la capa de entidad de dominio
Fuente: Elaboración Propia

Para crear entidades de dominio:

Arrastre la herramienta *DomainEntity* desde el Toolbox hacia el área del diagrama, se crea una forma para especificar una entidad.

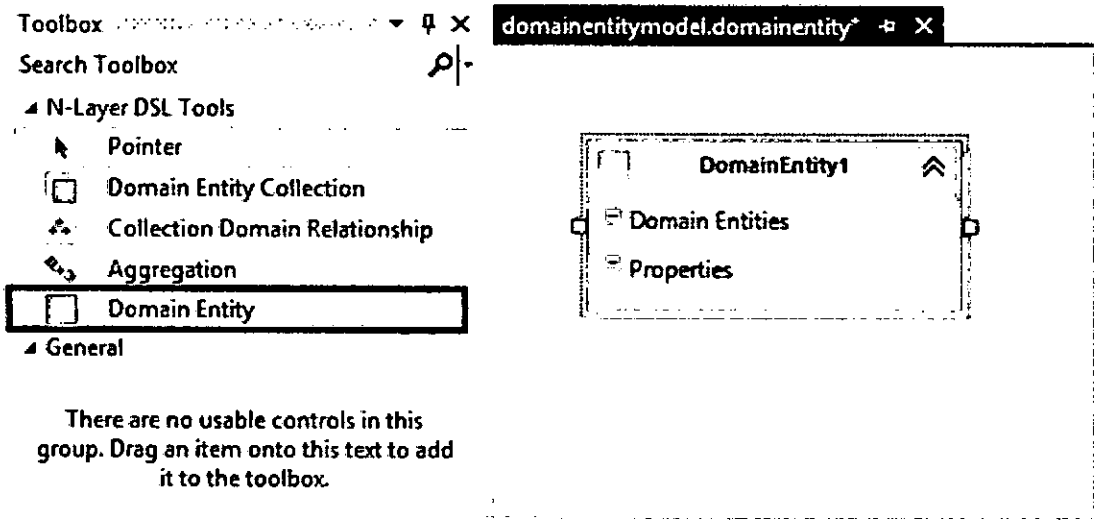


Figura 12.1.11. Agregar una entidad de dominio
Fuente: Elaboración Propia

Establecer propiedades de la entidad de dominio:

Para establecer las propiedades de la entidad de dominio seleccione la entidad de dominio y presione F4. Aparecerá la ventana de propiedades.

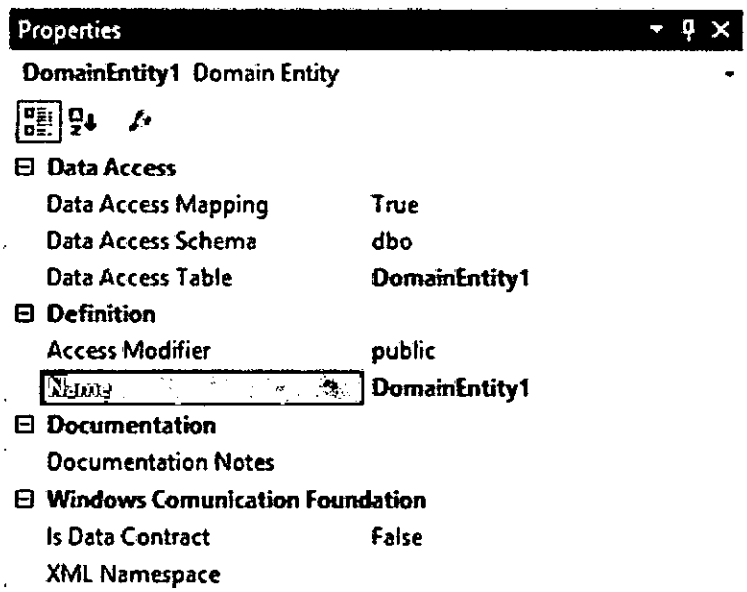


Figura 12.1.12. Propiedades una entidad de dominio
Fuente: Elaboración Propia

Las secciones de las propiedades de una entidad de dominio son Data Accces, para establecer las propiedades de acceso a datos; Definition, para establecer las propiedades de definición de la clase; Documentación para agregar código de documentación (comentarios de cabecera de clase entidad) y WCF que para este caso no se utiliza.

Agregar Propiedades primitivas:

Se pueden agregar propiedades primitivas de tipo String, Int32, etc. A una entidad de dominio para ello haga click derecho sobre la entidad y seleccione Add PrimitiveProperty.

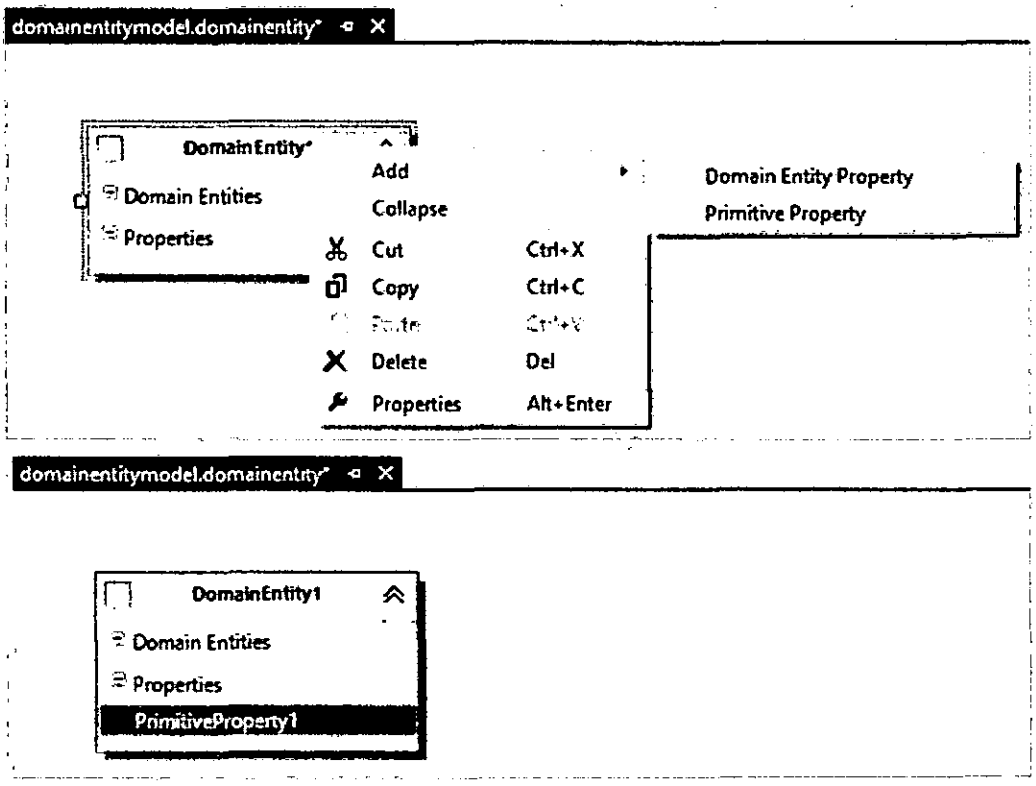


Figura 12.1.13. Agregar una propiedad primitiva a una entidad de dominio
Fuente: Elaboración Propia

Establecer propiedades de la propiedad primitiva de la entidad de dominio:

Para establecer las propiedades de la propiedad primitiva de dominio, seleccione la propiedad en cuestión y presione F4, se abrirá la ventana de propiedades.

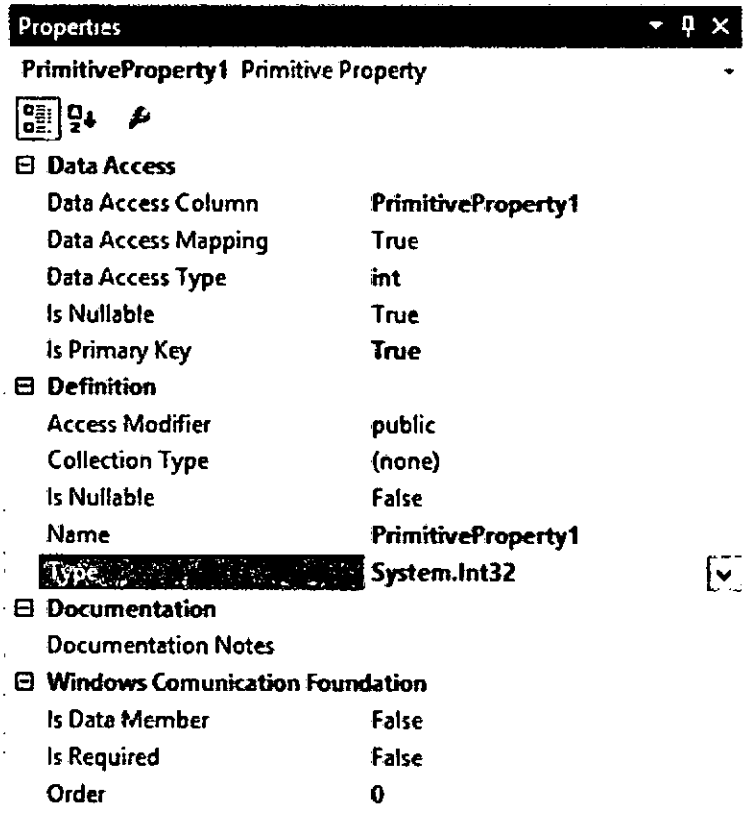
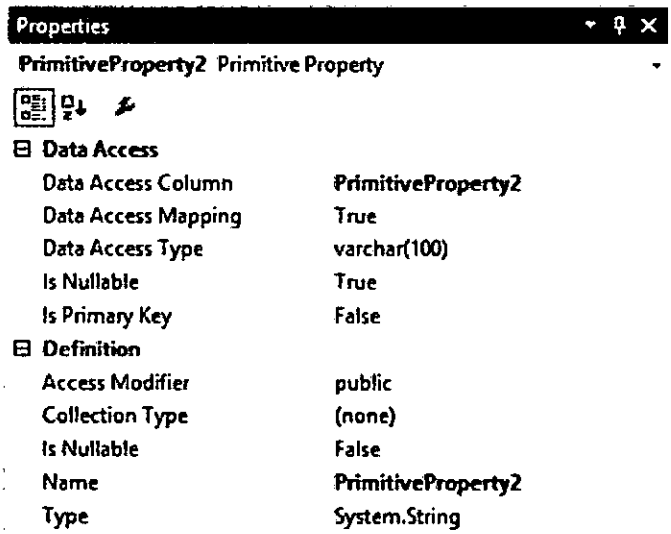


Figura 12.1.14. Establecer propiedades a una propiedad primitiva
Fuente: Elaboración Propia

Establecer las propiedades de la propiedad de dominio en las secciones DataAcces, que sirve para establecer propiedades de acceso a datos; Definition (definición de propiedad), Documentation (comentarios) y Windows Comunication Foundation, para este caso establecer las propiedades Is *Primary Key* en *true*, *Data Access Type* en *int* y *Type* en *System.Int32*. Como se muestra en la figura anterior.

Agregar dos propiedades más con la siguiente configuración.



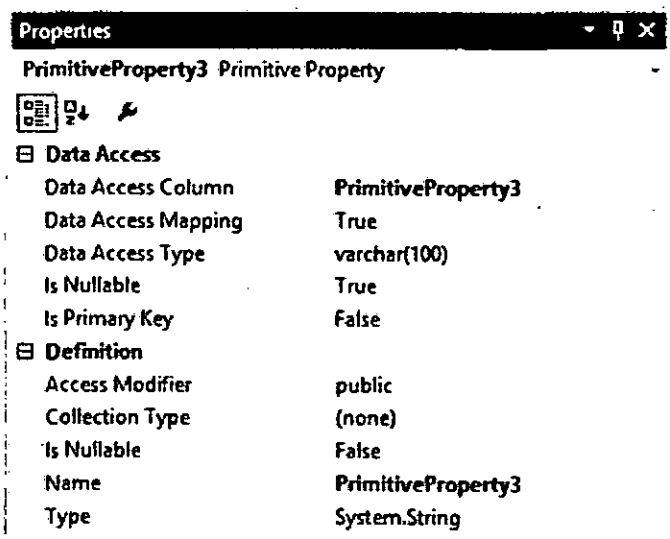


Figura 12.1.15. Configurar propiedades primitivas
Fuente: Elaboración Propia

Agregar otra entidad de dominio con la misma configuración de la entidad anterior pero con el nombre *DomainEntity2*.

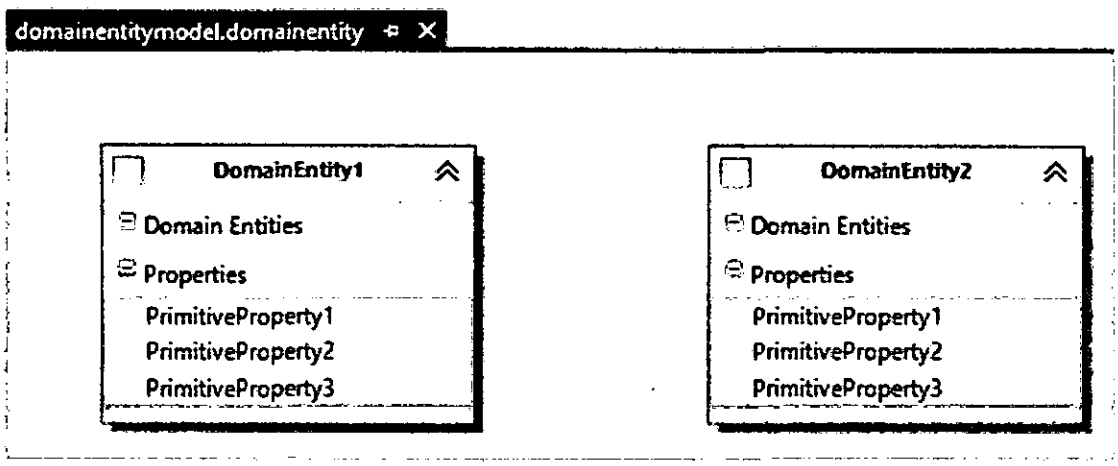


Figura 12.1.16. Agregar entidades de dominio
Fuente: Elaboración Propia

Agregar propiedades de entidad de dominio:

Se pueden agregar propiedades de entidad de dominio de dos formas:

La primera es agregándola directamente sobre la entidad, para ello haga click derecho sobre la entidad, luego seleccione Add y finalmente seleccione Domain Entity Property como se muestra en la figura 12.1.13.

Cuando se haya agregado se tendrá que definir el tipo de entidad de dominio a la que pertenece, para ello se tendrá que establecer la propiedad *Domain Entity* (véase la figura 12.1.14).

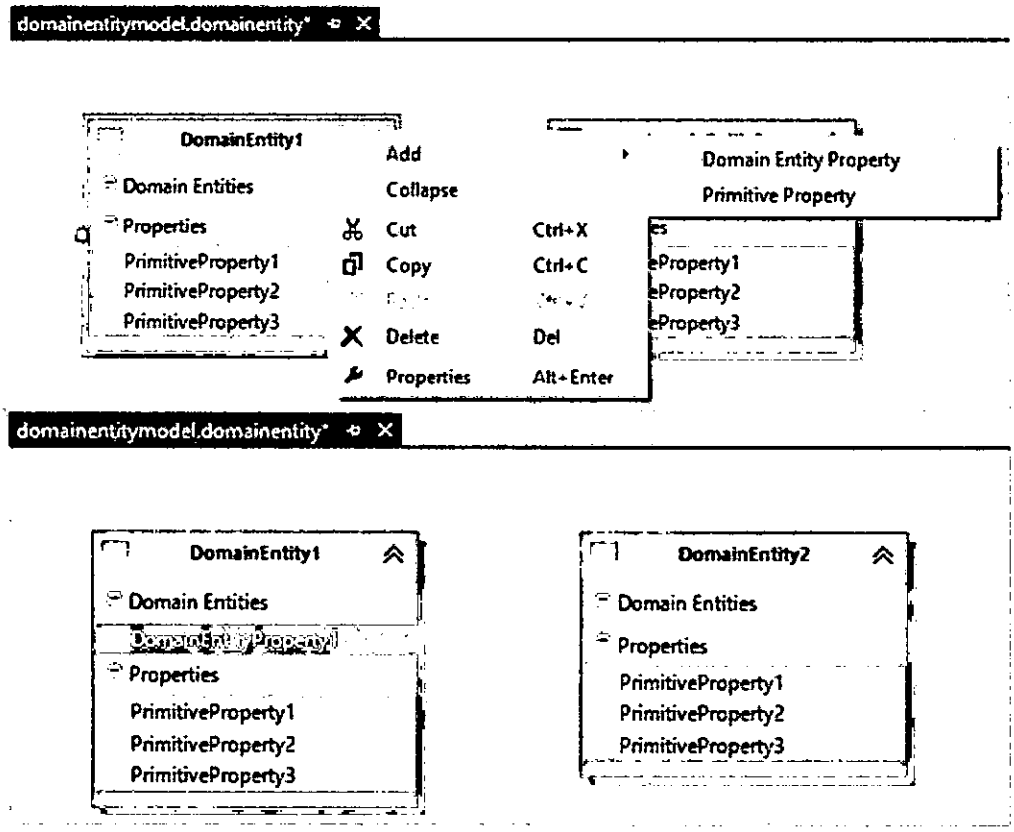


Figura 12.1.17. Agregar propiedades de entidad de dominio
Fuente: Elaboración Propia

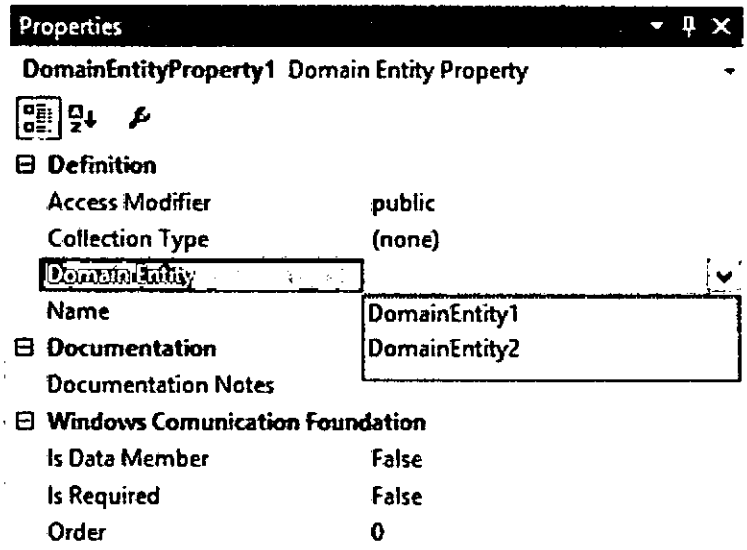


Figura 12.1.18. Establecer el tipo de entidad de dominio para una propiedad de dominio
Fuente: Elaboración Propia

Seleccione DomainEntity2. Verá cómo se crea una relación de agregación entre ambas entidades.

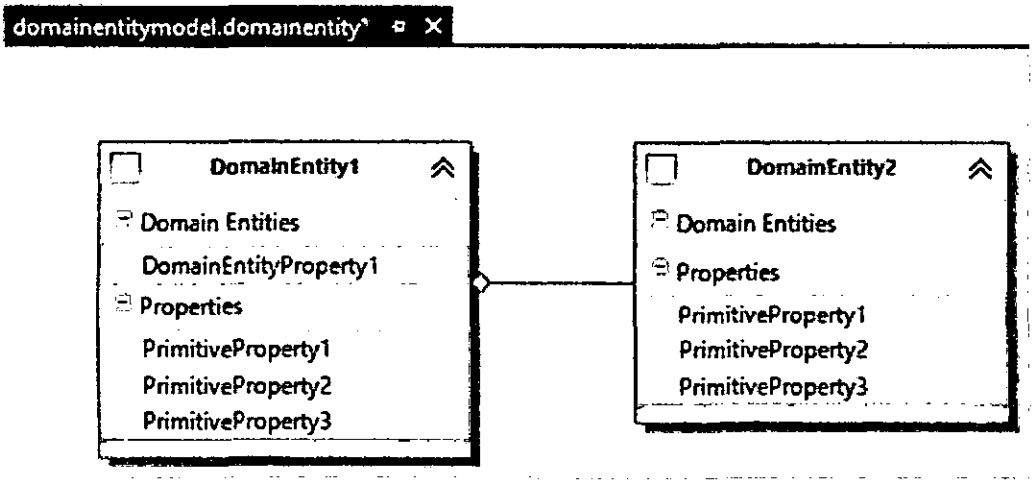


Figura 12.1.19. Relación de agregación entre entidades
Fuente: Elaboración Propia

La otra forma de agregar propiedades de entidad de dominio es usando directamente el conector de relación de agregación de entidades, este conector se encuentra en la barra de herramientas (Toolbox).

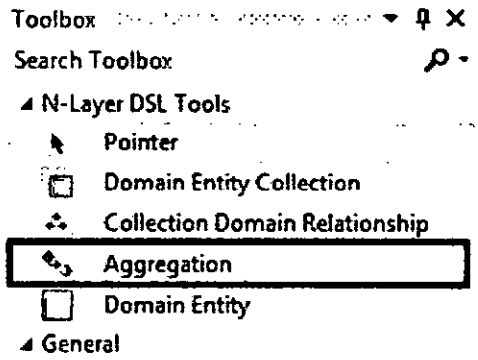


Figura 12.1.20. Conector de agregación de entidades
Fuente: Elaboración Propia

Partiendo del modelo de la figura 12.1.13, seleccione el conector de la barra de herramientas, luego selecciones la entidad DomainEntity2 y luego la entidad DomainEntity1, se tendrá un resultado como el de la figura siguiente:

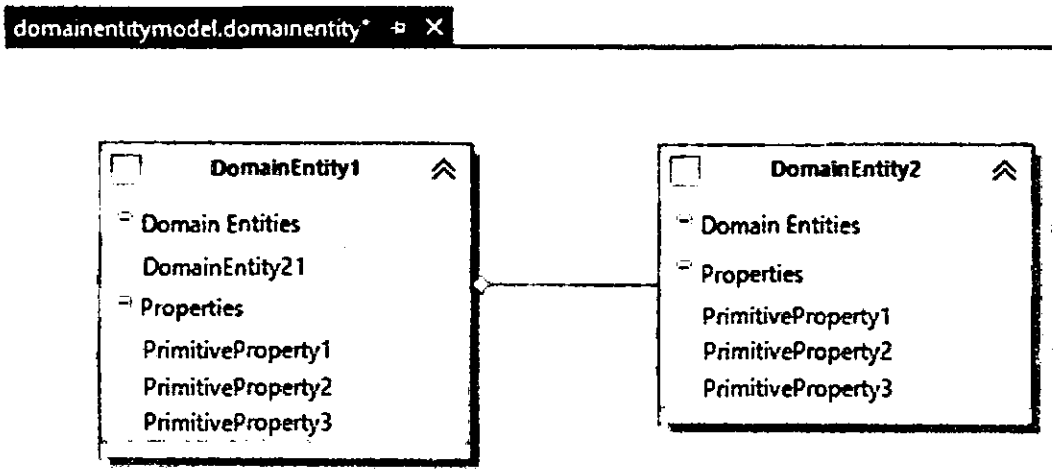


Figura 12.1.21. Relación de agregación entre entidades mediante conector
Fuente: Elaboración Propia

Agregar Colecciones de entidad de dominio:

Para agregar colecciones de dominio arrastre la herramienta *Domain Entity Collection* desde la caja de herramientas:

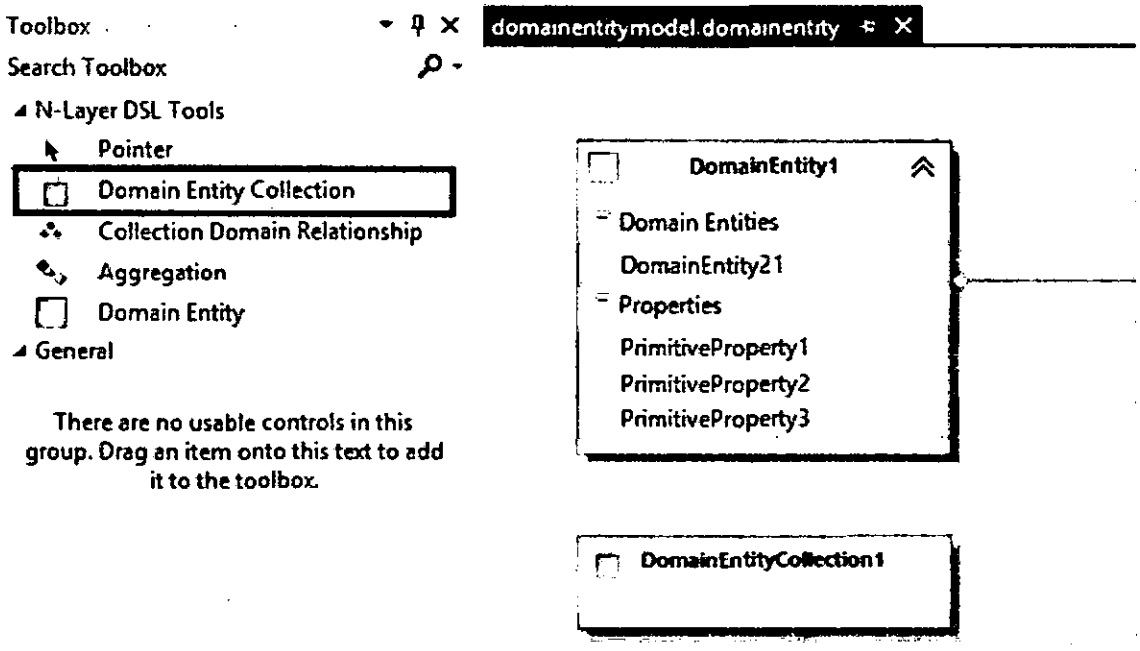


Figura 12.1.22. Relación de agregación entre entidades mediante conector
Fuente: Elaboración Propia

Se tiene que establecer el tipo de entidad de dominio para la lista de entidades de dominio, esto se puede hacer de dos formas:

La primera es editando la propiedad *Domain Entity Type* de la colección de entidades de dominio como se muestra en la figura siguiente:

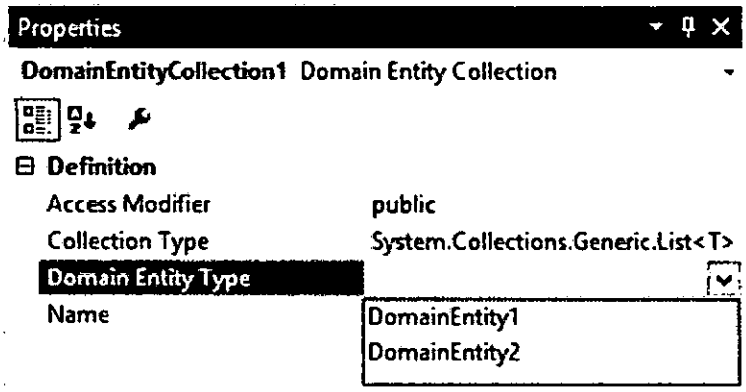


Figura 12.1.23. Establecer el tipo de entidad para la colección de entidades de dominio
Fuente: Elaboración Propia

Seleccione DomainEntiy1. Verá cómo se crea una relación de referencia entre la entidad de dominio DomainEntity1 y la colección de entidades de dominio DomainEntityCollection1.

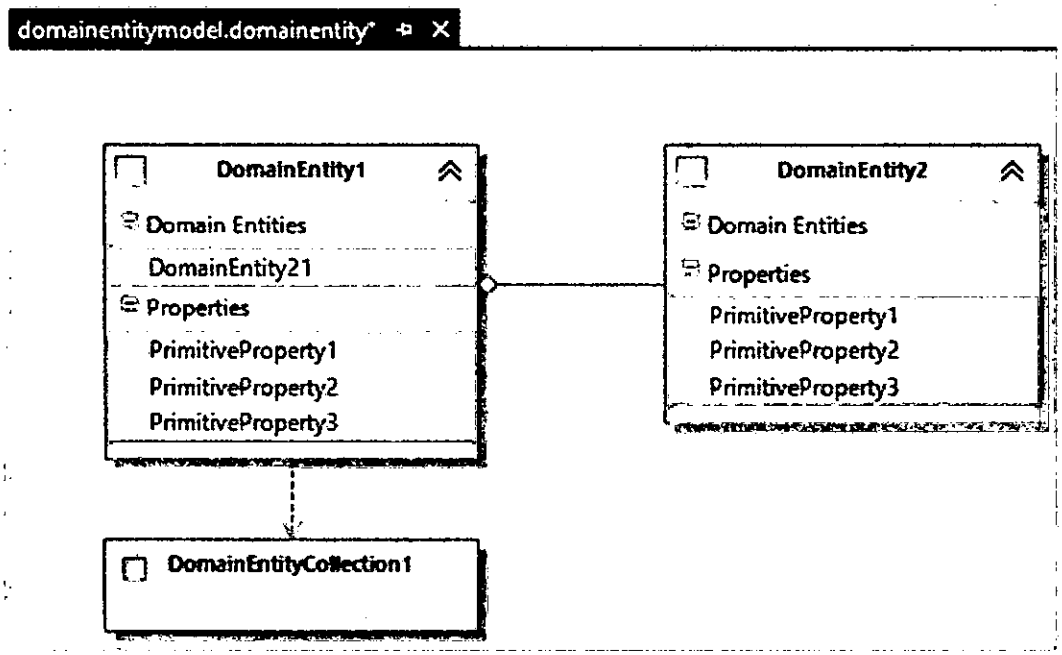


Figura 12.1.24. Relación de referencia entre una entidad y una colección de entidades
Fuente: Elaboración Propia

Otra forma de establecer la propiedad de referencia para especificar la entidad de dominio tipo para la colección de entidades de dominio, es usando la herramienta conector *Collection Domain Relashionship* de la caja de herramientas.

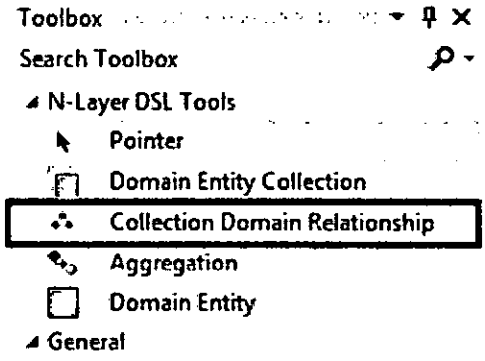


Figura 12.1.25. Conector de referencia de tipo de entidad para colección de entidades
Fuente: Elaboración Propia

Partiendo del diagrama de entidades de la figura 12.1.18, seleccione la herramienta mencionada y en primer lugar seleccione la entidad de dominio DomainEntity1 y luego la colección de entidades DomainEntityCollection1, vera como se crea la relación de referencia, dando como resultado el mismo diagrama de la figura 12.1.20.

Agregue una colección de entidades adicional y relaciónela con la entidad de dominio DomainEntity2, el diagrama quedaría de la siguiente forma:

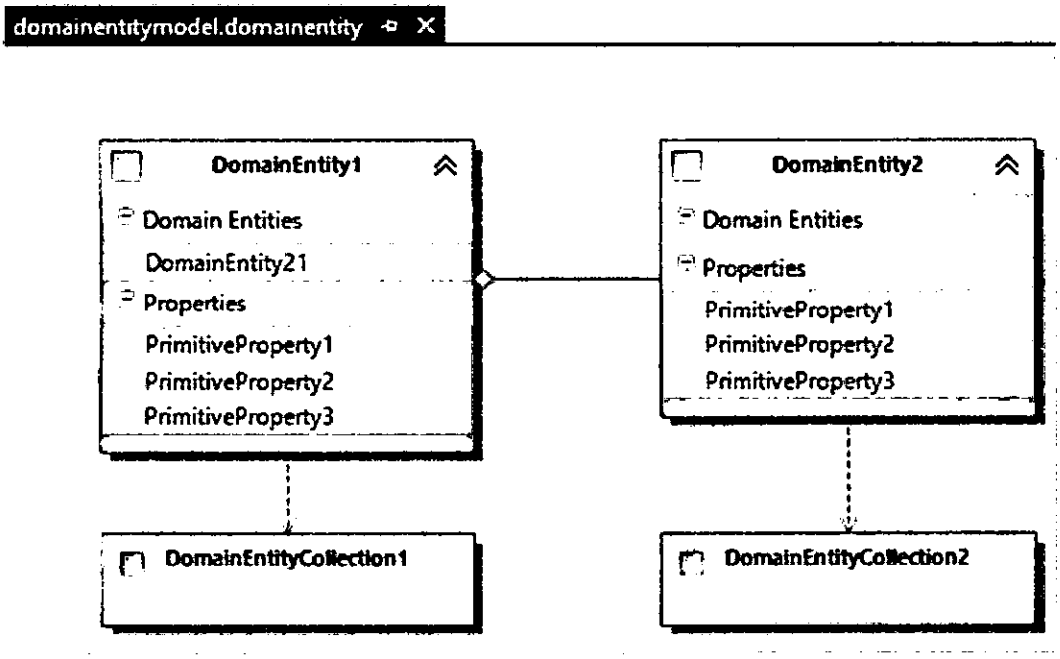


Figura 12.1.26. Modelo de entidades de dominio
Fuente: Elaboración Propia

4. GENERACIÓN DE CÓDIGO

Para generar código siga los siguientes pasos:

Paso 01: Del diagrama del ejemplo, haga click en una parte en blanco del diagrama y presión e F4 para establecer las propiedades del mismo.

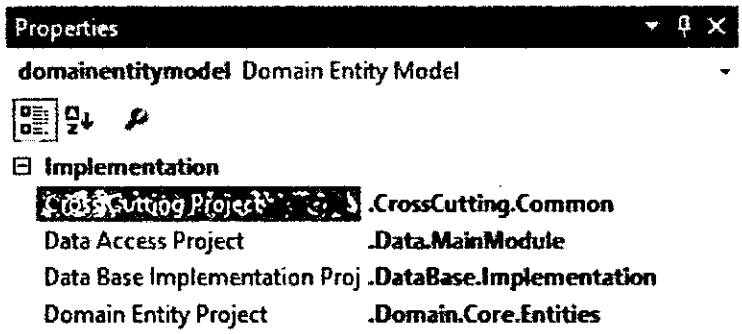


Figura 12.1.27. Propiedades del diagrama de entidades de dominio
Fuente: Elaboración Propia

Paso 02: Establezca las propiedades del dagrama de la siguiente manera:

Cross Cutting Project: Establecer el nombre del proyecto común de infraestructura transversal, este proyecto se encuentra en la carpeta *1 Layers → 1.5 Infraestructure → 1.5.2 Cross Cutting* y tiene el formato de nombre de acuerdo al nombre de la solución *[NombreSolucion].CrossCutting.Common.*

Data Access Project: Establecer nombre del proyecto de acceso a datos o proyecto de infraestructura de persistencia de datos, este proyecto se encuentra en la carpeta *1 Layers → 1.5 Infraestructure → 1.5.1 Data* y tiene el formato de nombre de acuerdo al nombre de la solución *[NombreSolucion].Data.MainModule.*

Data Base Implementation Project: Establecer nombre del proyecto de implementación de base de datos, este proyecto se encuentra en la carpeta *2 Database* y tiene el formato de nombre de acuerdo al nombre de la solución *[NombreSolucion].DataBase.Implementation.*

Domain Entity Project: Establecer nombre del proyecto de entidades de la capa de dominio, este proyecto se encuentra en la carpeta *1 Layers → 1.4 Domain → 1.4.1 Core* y tiene el formato de nombre de acuerdo al nombre de la solución *[NombreSolucion].Domain.Core.Entities.*

Establecidas los proyectos de generación de código, las propiedades del diagrama deberían quedar de la siguiente manera para el ejemplo:

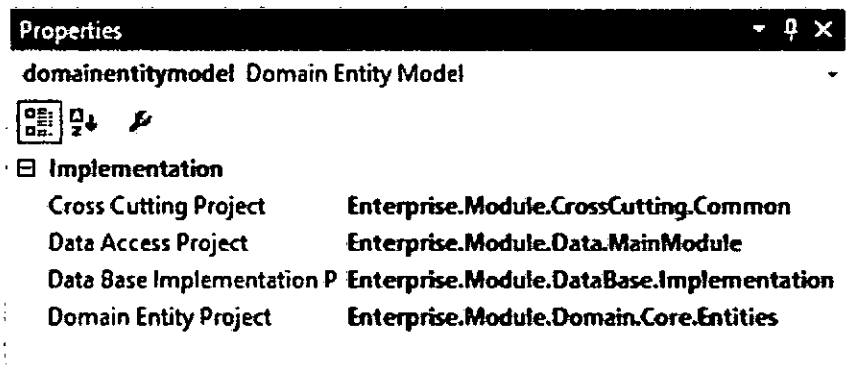


Figura 12.1.28. Establecer propiedades del diagrama de entidades de dominio
Fuente: Elaboración Propia

Paso 03: Para generar código, sitúese sobre una parte en blanco del diagrama, haga click derecho y seleccione generar código:

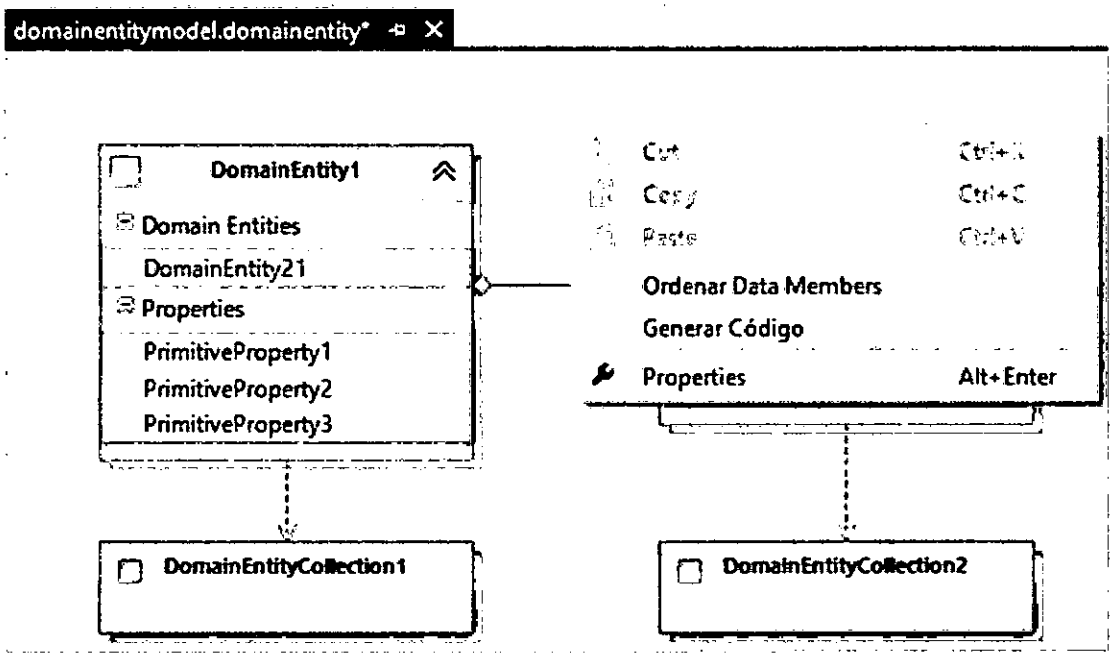


Figura 12.1.29. Generar código para las capas de la arquitectura
Fuente: Elaboración Propia

Para verificar que se ha generado código verifique cada uno de los proyectos configurados en las propiedades del diagrama establecidos en la figura 12.1.23.

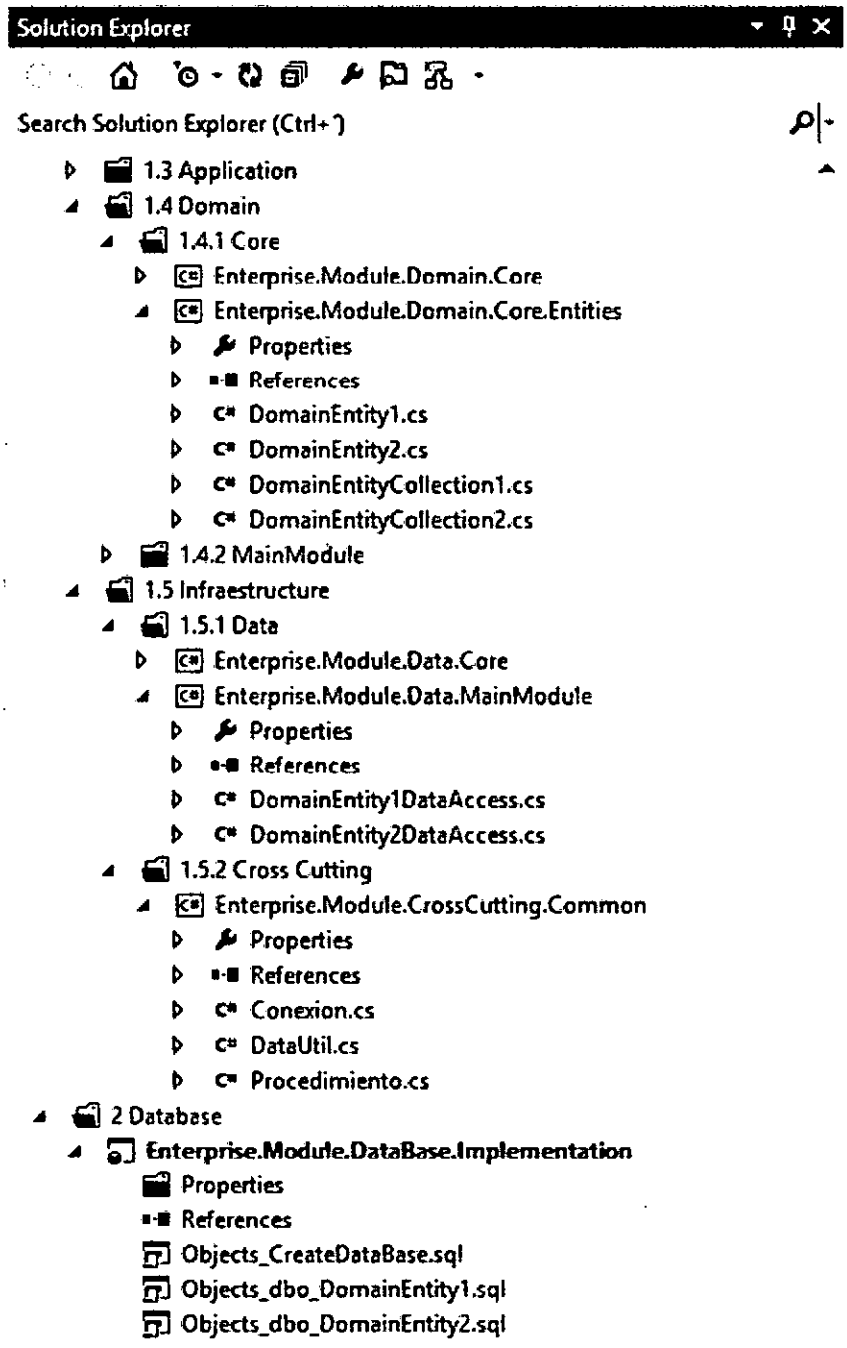


Figura 12.1.30. Código generado para el diagrama
Fuente: Elaboración Propia